UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior - Leganés

INGENIERÍA DE TELECOMUNICACIÓN



Proyecto Fin de Carrera

# Creación y Validación de un Cluster de Computación Científica Basado en Rocks


AUTOR: Ignacio Martínez Fernández

TUTOR: Luis Emilio García Castillo



Leganés, 2009

Título: *Creación y Validación de un Cluster de de Computación Científica basado en Rocks.*

Autor: Ignacio Martínez Fernández

Tutor: Luis E. García Castillo

La defensa del presente Proyecto Fin de Carrera se realizó el día 23 de Julio de 2009; siendo calificada por el siguiente tribunal:

Presidente: Harold Yesid Molina Bulla

Secretario Alejandro García Lampérez

Vocal Raúl Durán Díaz

Habiendo obtenido la siguiente calificación:

Calificación:

**Presidente** **Secretario** **Vocal**

*A mis padres*
*A mi hermana*
*A mis amigos*

*Ves cosas y dices, "¿Por qué? "Pero yo sueño cosas que nunca fueron y digo, "¿Por qué no? ".*
George Bernard Shaw

*Si tu intención es describir la verdad, hazlo con sencillez y la elegancia déjasela al sastre.*
Albert Einstein

*Problems?*
*In windows: reboot.*
*In linux: be root.*

## Agradecimientos

A mis padres que infundieron en mi los principios que rigen mi vida.

Gracias a mis amigos por la compresión y la confianza que han depositado en mi todos estos años.

A mi tutor por creer en mi y ayudarme a seguir creciendo intelectualmente.

Gracias a todos.
Ignacio

# Resumen

Cada vez más, resolvemos problemas cuya aproximación analítica es compleja, por lo que recurrimos a aproximaciones numéricas que reducen el problema a uno algebraico.

Dicha representación algebraica suele ser computacionalmente costosa de resolver tanto en tiempo, como en memoria. Es, por tanto, deseable aumentar la capacidad computacional del sistema con el fin de poder analizar estos problemas.

El desarrollo de la supercomputación ha propiciado y popularizado los clusters, facilitando la resolución de los problemas mencionados. Es objeto del presente Proyecto Fin de Carrera la creación y validación de un cluster de computación científica basado en Rocks.

# Abstract

Increasingly, we solve problems whose analytical approach is complex, that's why we use numerical approximations reducing the problem to an algebraic one.

This algebraic representation is often computationally expensive to resolve in time, as in memory. Therefore is desirable to increase the computing capacity of the system in order to be able to analyze these problems.

The development of supercomputing has facilitated and popularized the clusters, facilitating the resolution of these issues. The purpose of this Final Proyect is creating and validating a scientific computing cluster based on Rocks.

# Índice general

# Índice de figuras

# Índice de ficheros

# Capítulo 1

# Introducción

## 1.1. Motivación y Antecedentes

Actualmente necesitamos, cada vez más, resolver problemas complejos como, por ejemplo, simulaciones meteorológicas, plegamiento de proteínas, caracterización electromagnética de circuitos y estructuras radiantes, etc. Dado que la solución analítica de las ecuaciones diferenciales o integro-diferenciales que modelan el comportamiento físico sólo es posible en geometrías simples y con excitaciones simples, los simuladores hacen uso de métodos numéricos para hallar una solución al problema, que, pese a no ser la solución exacta, constituye una muy buena aproximación a ésta. La simulación numérica se encarga, a grandes rasgos, de facilitar el modelado del problema, de resolverlo en un entorno de precisión finita, y posteriormente de presentar los resultados. Esta formulación numérica del problema es muy flexible ante cambios de geometría, por lo que el mismo simulador es capaz de resolver el mismo problema físico aplicado a distintas estructuras.

Los diferentes métodos numéricos se distinguen por la forma en que proyectan el conjunto de ecuaciones diferenciales o integro-diferenciales en un espacio de dimensión finita, discretizando el problema. Lo que tienen en común todos ellos es que al final del proceso de discretización se llega a un sistema algebraico de ecuaciones $[A]\mathbf{x} = \mathbf{b}$ que debe resolverse. Las diversas formas de discretización se reflejan en la estructura de la matriz $[A]$: densa o dispersa, definida positiva o no, etc. La solución del sistema algebraico de ecuaciones permite obtener la incógnita (variable) del problema físico (presión, temperatura, campo electromagnético, etc) y, a partir de esta vía con diversos post-procesos, podemos obtener los diferentes parámetros de interés. En este contexto, hay que destacar el trabajo de

investigación de diversos profesores del Grupo de Radiofrecuencia del Departamento de Teoría de la Señal y Comunicaciones en electromagnetismo computacional en el ámbito de la simulación numérica de problemas electromagnéticos, y más específicamente, en problemas de electrodinámica de alta frecuencia. Es precisamente en el seno de dicho Grupo en el que se ha desarrollado el presente Proyecto Fin de Carrera.

Los problemas a los que se ha hecho referencia anteriormente se caracterizan por dar lugar a matrices del sistema $[A]$ muy grandes, tanto debido a la complejidad de la geometría y/o configuración de materiales de la estructura bajo análisis, como por el tamaño de ésta. El tamaño de la estructura bajo análisis debe entenderse en términos relativos a las longitudes de onda de las diversas ondas que puedan propagarse en la estructura. Por ejemplo, en electromagnetismo de alta frecuencia se consideraría el "tamaño eléctrico" de la estructura como el tamaño físico relativo a la longitud de onda menor (de componente frecuencial mayor) presente en el problema. De este modo, la simulación numérica requiere una elevada capacidad de cómputo. La capacidad de cómputo debe entenderse en su doble vertiente de velocidad de las operaciones numéricas (operaciones en coma flotante) y de la memoria disponible. La velocidad determina el tiempo que se necesita para disponer de los resultados.

Este tiempo debe ser lo suficientemente reducido para que sea práctico acometer la simulación del problema. Por otro lado, la memoria determina si un problema puede resolverse o no. En este sentido, conviene aclarar que por memoria nos referimos típicamente a la memoria RAM, dado que el almacenamiento en disco tiene tiempos de acceso mucho mayores que la memoria RAM y su uso no resulta, en general, práctico. En este punto conviene hacer mención al desarrollo de algoritmos específicos para uso de memoria de disco (denominados algoritmos *out of core*, que bien pueden ser paralelos), aunque en el presente Proyecto Fin de Carrera no hemos hecho especial hincapié en esta familia de algoritmos.

Por todo lo anteriormente expuesto, es deseable aumentar la capacidad computacional del sistema con objeto de poder analizar problemas "mayores" y/o más complejos; análisis que es demandado por el tejido industrial y también por los desarrollos llevados a cabo dentro de las diversas líneas de investigación del Grupo.

En la última década un gran desarrollo en supercomputación ha hecho posible pasar de sistemas muy especializados de memoria compartida sólo a disposición de centros de supercomputación, a clusters (sistemas de memoria distribuida) combinando ordenadores "convencionales" ya a disposición de cualquier grupo de trabajo.

Conviene notar que la creación de un cluster no es solamente la mera adquisición del hardware que lo compone, sino que además de aspectos que podemos denominar igualmente hardware, como son la refrigeración y la alimentación eléctrica, un aspecto primordial es la organización software del cluster, su mantenimiento, y las labores de administración asociadas. En este sentido, uno de los objetivos será la minimización de las tareas de administración y mantenimiento ayudándonos de Rocks como veremos a lo largo del desarrollo del presente Proyecto Fin de Carrera.

## 1.2. Objetivos

El objetivo del presente Proyecto Fin de Carrera es la creación, prueba y validación de un cluster de cálculo científico.

El objetivo podemos desglosarlo en los siguientes subobjetivos o tareas:

- Creación de un cluster de cálculo científico.

  El objetivo aquí es la creación de un cluster HPC (*High Performance Computing*) capaz de ejecutar algoritmos paralelos de una forma eficiente.

  Como hemos señalado, necesitaremos un entorno estable, escalable en la medida de lo posible, y que facilite su configuración y mantenimiento.

  Para lograr todos estos puntos usamos Rocks, una distribución orientada a minimizar los tiempos de configuración.

- Comprobar el correcto funcionamiento del cluster ejecutando un benchmark de rendimiento.

- Probar algoritmos en paralelo, con el fin de comprobar el funcionamiento del cluster bajo condiciones reales:

  - Comenzaremos desarrollando un resolvedor paralelo iterativo, principalmente pensado para matrices dispersas, con la ayuda de *PETSc*.
  - Utilizaremos un resolvedor multifrontal paralelo para matrices dispersas generadas por el Método de Elementos Finitos.
  - Por último utilizaremos un resolvedor paralelo de matrices densas, generado por MoM (*Method of Moments* [1]).

## 1.3. Organización de la memoria

En el segundo capítulo trataremos la creación de un cluster de computación científica. Un lector familiarizado con los clusters podrá evitar la lectura de la primera sección. Después analizaremos el hardware disponible y describiremos la distribución utilizada, así como los motivos que nos han llevado a utilizarla. Para concluir el capítulo se expondrán algunas funcionalidades del cluster.

El tercer capítulo desarrolla cómo añadir y configurar software de diversas formas, culminando con cómo crear discos de instalación y configuración en Rocks (Rolls).

En el capítulo cuarto exponemos los resultados obtenidos resolviendo problemas "reales", para cuya resolución emplearemos diferentes métodos. El primero de los métodos de resolución será un método iterativo paralelo, que desarrollaremos, pensado para tratar matrices dispersas. Un lector familiarizado con comunicaciones entre diferentes lenguajes podrá omitir la lectura del interfaz C/Fortran. El segundo método de resolución será un método multifrontal paralelo para matrices dispersas, y por último, utilizaremos un programa cuyo resolvedor paralelo trabaja sobre matrices densas.

El quinto y último capítulo presenta las conclusiones extraídas del trabajo realizado y líneas futuras de trabajo que mejorarían el presente proyecto.

Los anexos recogen con mayor profundidad determinados puntos del proyecto. El primer anexo versa sobre métodos matemáticos a los que hacemos referencia en el desarrollo del resolvedor iterativo.

En el segundo anexo plasmamos los pasos necesarios para la instalación de Rocks e incluye algunos consejos extraídos del desarrollo del proyecto.

El anexo tercero es un resumen de uso de Sun Grid Engine [2], extraída del Wiki realizado en el desarrollo del proyecto.

El cuarto anexo expone brevemente cómo *modules* [3] modifica las variables de entorno.

El último anexo presentamos la Roll Restore, un sistema de backup de estado propio de Rocks, con un ejemplo de uso.

# Capítulo 2

# Creación del cluster

## 2.1. ¿Qué es un cluster?

Como definición, podemos tomar que un cluster es un conjunto de ordenadores susceptibles de trabajar en una tarea común. Como veremos, sólo necesitaremos estar en el nodo maestro, o frontend, para poder hacer pleno uso de todo el conjunto. Se asemeja a la idea del principio de divide y vencerás, aplicándolo sobre los procesos que están ejecutándose en el ordenador.

Primero veamos una clasificación de los clusters:

- Clusters de alta disponibilidad (High Availability)
- Clusters de balanceo de carga (Load Balancing)
- Clusters de alto rendimiento (High Performance Computer)

También decir que pueden ser homogéneos, cuando todos los ordenadores tienen el mismo hardware y software, o heterogéneos, cuando éstos difieren.

Los clusters de alta disponibilidad son aquellos que ante fallos de hardware, son capaces de seguir funcionando. Se basan en redundancia; esto es, que poseen uno o más nodos que pueden realizar las tareas de otro de los nodos, en caso de que deje de funcionar. De esta manera serán capaces de continuar funcionando ante indisponibilidades de hardware. Se incluyen aquí los clusters de almacenamiento, aunque pueden presentar esquemas de aumento de la capacidad, la característica

principal es la redundancia. Un claro ejemplo sería un *datacenter*, o un servidor de backups.

Los clusters de balanceo de carga, son una variante de los anteriores, pero en lugar de tener redundancia, poseen la característica de reducir la latencia y aumentar la capacidad. Están configurados para dividir el trabajo, por lo que en caso de que uno falle o deje de funcionar, cualquiera de los otros pueden asumir las funciones del que falla. Una aplicación de este tipo de clusters sería un servidor web.

En nuestro caso, nos centraremos en los clusters de alto rendimiento. Necesitamos potencia de cálculo y esta necesidad es prioritaria frente a la disponibilidad del cluster, aunque no la perderemos de vista. Las capacidades de disponibilidad y fiabilidad las implementaremos por software. Debemos aclarar que no estamos ante un HTC (*High Throughput Computing*, un cluster de balanceo de carga sin redundancia), es un HPC, no tenemos varios ordenadores trabajando a la vez en problemas diferentes, la aproximación es un conjunto de nodos trabajando simultáneamente en la misma tarea. Trataremos de ser más claros: un cluster HTC consistiría en varios usuarios mandando tareas hasta completar la capacidad del cluster, mientras que un HPC sería un sólo usuario mandando una tarea que ocuparía toda esta capacidad. En un cluster HPC podemos enviar trabajos que se desglosen en varios procesos ligados, que compartirán estructura y comunicaciones, permitiéndonos abordar problemas que ninguno de los ordenadores, que componen el cluster, podría por separado.

## 2.2. Hardware disponible

Disponemos de los siguientes equipos:

- Cuatro Intel Xeon 64 bits Quad Core, biprocesadores con 32 Gb de RAM y dos interfaces de red Gigabit-Ethernet.
- Dos Intel Xeon 64 bits Dual Core, biprocesadores con 32 Gb de memoria RAM y dos interfaces físicos Gigabit-Ethernet.
- Siete Intel Xeon 64 bits, biprocesadores con 6/8 Gb de memoria RAM y dos interfaces físicos Gigabit-Ethernet.
- Switch Gigabit-Ethernet.

Figura 2.1: Hardware inicial.

Al inicio del presente proyecto disponíamos de la mayor parte del hardware listado (figura 2.1), salvo los cuatro nodos Intel Xeon 64 bits Quad Core, que fueron adquiridos durante el mismo.

Durante el proyecto se realizaron algunas ampliaciones, como por ejemplo de memoria RAM (figura 2.2).

Y al final después de incorporar todos los nodos a un *rack* el resultado fue el que podemos ver en las figuras 2.3 y 2.4.

En total dispondremos de unos 54 cores y cada uno de ellos nos aportara entre 12 y 60 Gflops. En conjunto estaremos entorno a los 480 Gflops, ya que alguno de los equipos se destinará a labores de administración. Y debemos aclarar que entendemos por nodo un ordenador independientemente del número de procesadores que tenga (que serán dos en nuestro caso), por procesador el chip que contiene uno o más cores, y por core entendemos una unidad de procesamiento básica de la arquitectura actual.

Emplearemos la arquitectura x86_64 (em64t), todos los nodos disponen de un mínimo de 3 Gb de RAM por core y la velocidad de la red será de 1 Gbps, pues la agregación de redes de baja latencia o de velocidades superiores encarece notoriamente el equipo, y fue descartada.

Figura 2.2: Ampliación de RAM.

Figura 2.3: Frontal del Cluter.

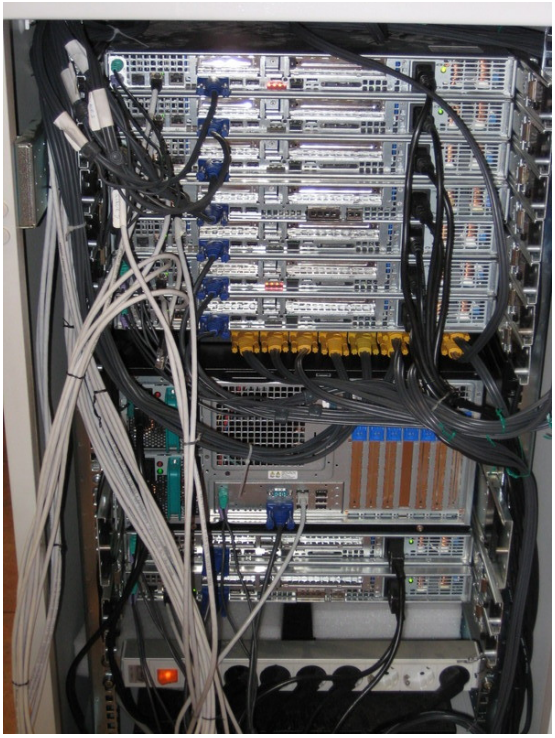


Figura 2.4: Vista posterior del Cluster.

La arquitectura del cluster puede ser uno de los puntos más importantes debido a que es uno de los elementos más críticos. Para ilustrar nuestra afirmación exponemos nuestro caso: la tasa de transferencia mínima en el FSB (*Front Side Bus*, bus del sistema) es de 4,25 Gbps y la máxima es de 21 Gbps, siendo mayor entre la memoria caché y la CPU. Como la máxima tasa de trasferencia de la red es de 1 Gbps, éste es el factor más limitante. Lo es más aún que la tasa máxima de transferencia del disco duro, de unos 800 Mbps (100 MBps), porque, aunque el disco duro se utiliza al inicializar el problema, con los datos de entrada y brevemente al término del mismo para escribir los resultados, no interviene (o no debe intervenir) en el cálculo; mientras que la red interviene en la distribución del problema y en los cálculos.

Debemos aclarar que cuando interviene el disco duro en los cálculos el algoritmo se denomina *out of core*, frente al algoritmo *in core* donde sólo interviene la memoria RAM, con el primero podemos tratar problemas mayores más lentamente, y al revés con el segundo. Nosotros trataremos de estar siempre *in core* y tratar problemas con un mayor número de incógnitas con la paralelización. Lo que no quiere decir que los resolvedores *out core* sean una mala opción, pues si el problema es muy grande constituyen una buena opción, y habría que reconsiderar este punto, optando, tal vez, por sistemas RAID que mejoren el rendimiento de

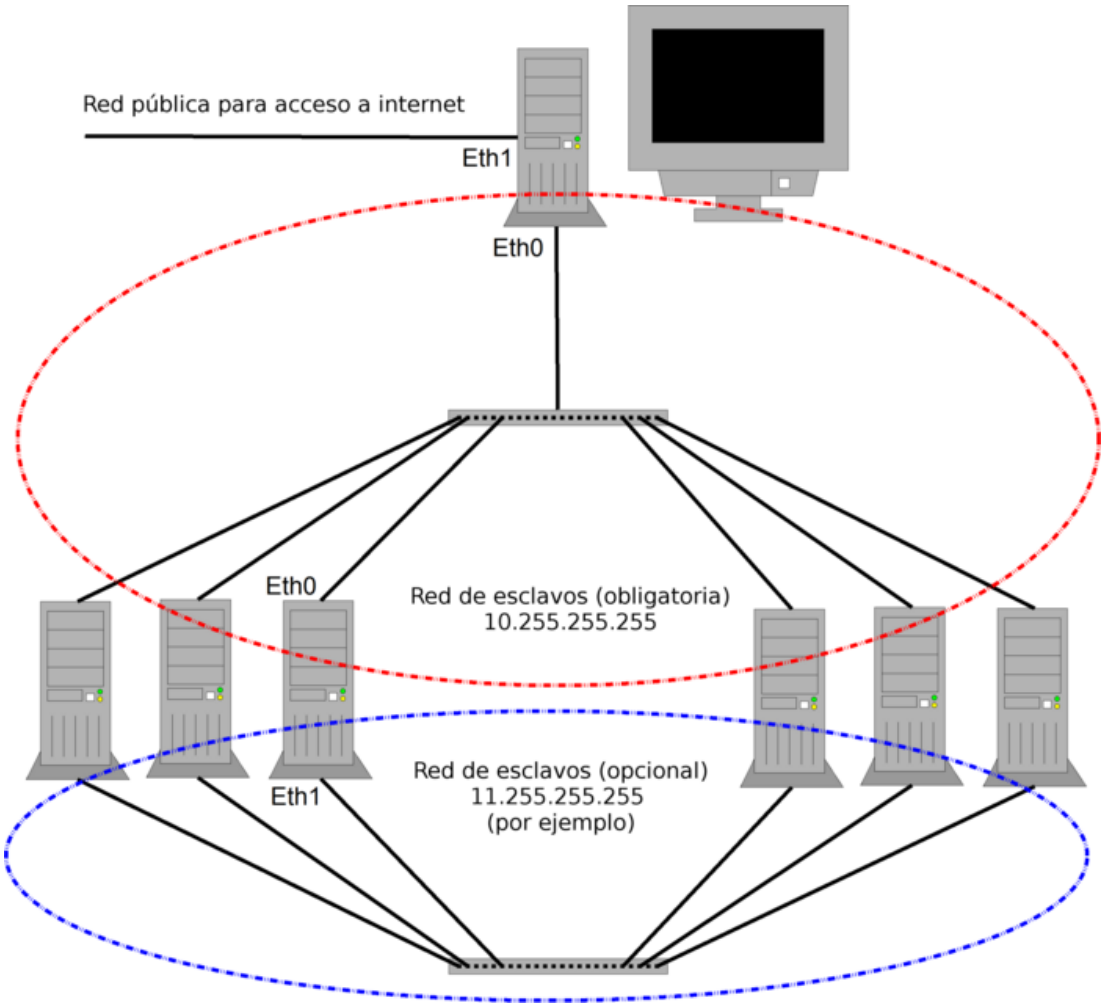



Figura 2.5: Esquema detallado de red.

lectura/escritura a disco.

Hay dos segmentos en el cluster que no deben tratarse por igual, el primero el segmento de conexión con el mundo exterior, que sólo ha de llegar al nodo maestro o frontend y el segmento que interconecta internamente el cluster, que en nuestro caso está compuesto por el frontend, los nodos y el switch. Si se necesitase ampliar la red, la topología en árbol resulta ser la indicada. Los nodos tendrán el mismo número de saltos al maestro y los dominios de colisión se separarán por medio de switches.

Que los nodos tengan el mismo número de saltos hasta el frontend, se traduce en un retardo similar, dado que todos los nodos (desde el punto de vista de cores),

disponen capacidades de computo similares, eliminamos en la medida de lo posible retardos o desincronizaciones innecesarias.

Al disponer de 2 interfaces de red, podríamos dedicar uno de ellos a gestionar el cluster, y el otro estará dedicado a procesos de cálculo, tal y como podemos ver en la figura 2.5. Pero a fecha de la finalización del proyecto, no se implementó la red de esclavos opcional, dedicada al cálculo, por lo que el tráfico de gestión y cálculo se cursa por la red obligatoria.

Siempre trataremos de minimizar la ocupación de la red, reduciendo el número de servicios que hacen uso de la red. Cada nodo contará con librerías propias instaladas localmente en el disco duro del nodo. Estas librerías estarán, eso sí, controladas desde el frontend. Sólo se empleará NFS (*Network File System*, podemos encontrarlo en [4] y [5]) en algunos casos para su distribución en la fase inicial de instalación (véase el Capítulo 3). De esta manera el tráfico de la red se destina exclusivamente labores de mantenimiento, monitorización y cálculo.

## 2.3. Rocks: motivación y capacidades.

Podemos hacer el ejercicio de calcular el precio de un cluster de tamaño medio, y cuanto se encarece al añadir cierto tipo de hardware, como redes de baja latencia (entorno al 33 %) o un KVM, *Keyboard Video Mouse*, (aproximadamente un 5 %), pero lo que de verdad encarece el cluster es la administración y la gestión. Un cluster no es sólo las máquinas que lo componen, necesita, al menos, de una persona que se encargue de solventar los problemas derivados del uso y mantenimiento de este, estos problemas podemos agruparlos en tres tipos:

A. Administración: necesitamos un sistema que guarde el estado del cluster y de los nodos de cálculo. Por estado de cluster nos referimos a la situación general del frontend, a cómo responde en un instante determinado. Siempre nos interesará mantener el frontend (y los nodos) en un estado estable. No nos interesará el estado de los nodos (salvo el inicial), porque al carecer de servicios y sólo aportar la capacidad de cálculo, son prescindibles. La monitorización del cluster es fundamental: necesitamos saber si un nodo ha dejado de funcionar, o diferenciar qué nodos están ejecutando y cuales no. Al disponer de esa información podremos realizar las tareas administrativas pertinentes eficazmente.

B. Aplicaciones: la congruencia de las configuraciones y librerías. Este aspecto es primordial para el correcto funcionamiento de cualquier aplicación paralela, y suele ser común tener varias versiones de la misma biblioteca.

   Con el fin de simplificar la congruencia del software, necesitaremos una arquitectura lo más homogénea posible. Además las librerías y paquetes que instalemos o pretendamos usar han de ser compatibles con la arquitectura (con todas ellas), es decir, si disponemos de Xeon de 64 bits, los nodos que agreguemos deberán ser Xeon de 64 bits; si por el contrario fuesen core i7 o pentium dual core podrían ocasionarse problemas, pues no poseen el mismo conjunto de instrucciones y al compilar lo haremos para una de las arquitecturas. En nuestro caso el único nodo capaz de compilar es el frontend, y el resto de nodos deberán poseer la misma arquitectura.

   Lo cierto es que no existe el cluster 100 % homogéneo, por el ciclo de vida que tienen, pero en la medida de lo posible debemos conservar la misma arquitectura.

C. Control de las tareas paralelas: necesitamos ejecutar tareas, eliminarlas, programarlas (en el tiempo), monitorizarlas, ... Necesitamos un conjunto de herramientas (gestores de colas y bibliotecas de paso de mensajes, básicamente) que nos den control sobre estas tareas de la forma más simple posible.

   Este punto es clave, porque la finalidad de un cluster es la ejecución de programas. Si carecemos de control, o es confuso, no podremos hacer un buen uso del cluster.

El middleware que se sitúa en cada lado de un sistema de computación distribuida, ver figura 2.6, funciona como una capa de abstracción sobre la que se pueden desarrollar aplicaciones sin importar que tengamos un número determinado de nodos, o que sean de diferentes arquitecturas, conformando el cluster. Además es independiente de los servicios de red, y tiene disponibilidad absoluta, sin importar las tecnologías de red que tengamos disponibles. Es fundamental para este tipo de sistemas, pues la complejidad de crear aplicaciones que se comuniquen por red, a un número indeterminado de máquinas de diferentes arquitecturas a través de redes heterogéneas, es extremadamente costoso, por no decir inviable.

Los problemas apuntados anteriormente, puntos de administración del estado, de control del software y de las tareas paralelas, podemos verlos reflejados en la figura 2.6, dado que la capa de *middleware* trata de resolver los problemas anteriormente expuestos. Las correspondencias serían:

- El problema A de administración del estado se trata en *Cluster State Management/Monitoring*.

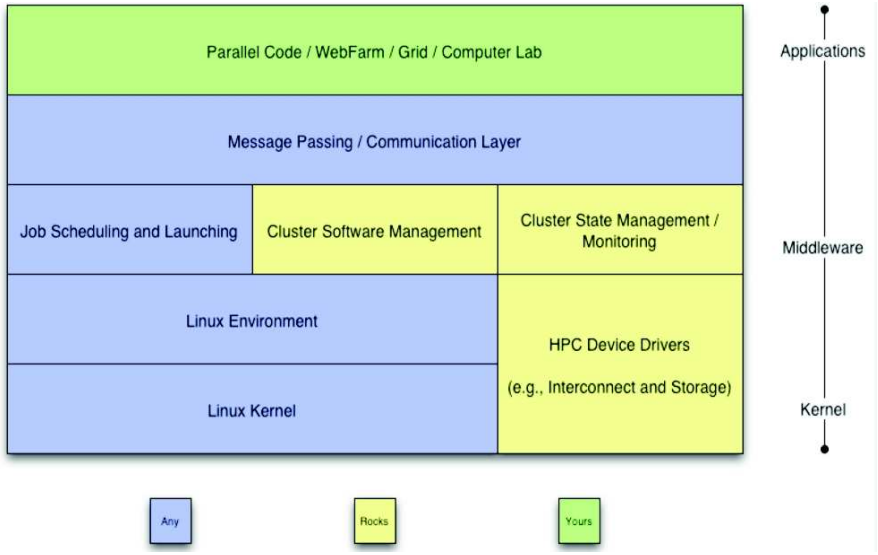



Figura 2.6: Localización del middleware.

- B, la congruencia del software y las variables de entorno, se solventa con *Cluster Software Management*, ayudado con *modules*.
- El punto C, de control de tareas paralelas, lo resolvemos con las capas de *Message Passing/Comunication Layer* y *Job Scheluding and Launching*.

  Hay que resaltar la importancia de la capa llamada Message Passing/Communication Layer. Como podemos observar cada uno de los equipos dispone de su propio sistema operativo y aplicaciones básicas a bajo nivel. Sin esta capa dispondríamos de un sistema de ejecución paralela de tareas (de múltiples tareas secuenciales). La capa de comunicación la realizaremos con MPI (Message Passing Interface) [6].

Ahora bien, para crear el cluster debemos agrupar los nodos por funcionalidad, siguiendo el esquema de la figura 2.5:

- Frontend: Es un sólo nodo, expuesto al mundo exterior y deberá proveer servicios a los nodos esclavos, con lo que será necesario disponer de dos interfaces Ethernet físicas: una para la red externa, que proveerá conexión con el mundo exterior, y otra para la interna, que es la red que comunica los nodos esclavos con el maestro. Desde este nodo, trataremos de realizar todo el mantenimiento, y la toma de decisiones. El nodo maestro debe realizar tareas de encaminamiento, y proveer los servicios básicos, de los que hablaremos posteriormente. El almacenamiento puede realizarse en el

nodo maestro o en un NAS (*Network Attached Storage*), pero jamas en los esclavos, porque, extendiendo la filosofía, éstos deberán ser prescindibles y sustituibles.

- Esclavos: Los esclavos no ven la red externa directamente, sólo a través del nodo maestro. Es extremadamente recomendable, por no decir imprescindible, que la secuencia de boot de los esclavos se inicie por red. Ya que en caso de existir un fallo, el sistema procederá a devolverlo a un estado estable, ya sea reiniciándolo o reinstalándolo. Siempre por red, es impensable tener que hacerlo a mano, salvo que el sistema tenga muy pocos nodos.

Basándonos en nuestras necesidades, y que queremos construir un cluster de alto rendimiento, que además tenga determinadas bibliotecas, buscamos en un listado de distribuciones orientadas a clusters HPC *High Performance Computing*, con aquellas que podrían servirnos. Esta discusión sólo tiene sentido, si buscamos una distribución que integre tanto middleware, como sistema operativo (como es nuestro caso). La búsqueda no fue exhaustiva y a continuación mostramos un listado de las distribuciones que más se acercaban a nuestros objetivos:

- Rocks [7] centra sus esfuerzos en crear un sistema escalable, heterogéneo y sencillo de usar, trata de evitar la dependencia de un administrador. Tiene una gran variedad de software. No es un sistema operativo, se instala sobre CentOS, o cualquier otro sistema operativo basado en RedHat Entreprise Level.

- Warewulf/Perceus [8]: se sirve de un Virtual Node File System para homogeneizar todos los nodos. Es parecido a Rocks, pero no incluye librerías y deberemos adaptarlas a la distribución. Se emplea junto con nodos sin disco duro propio.

- SCYLD Beowulf [9]: tiene problemas de escalabilidad, debido a que parte del proceso permanece en el maestro, es decir parte del problema se ejecuta siempre en el maestro, lo que genera problemas de memoria, actualmente es comercial. Nosotros preferimos herramientas de código libre.

- Score[10]: la instalación en los nodos esta semi-automatizada, cuenta con un buen gestor de procesos y protocolos propios para Myrinet. La programación paralela se realiza con plantillas. No es un SO como tal es más parecido a una capa de middleware, que se monta sobre CentOS y que llega hasta los interfaces de red. Rocks esta desarrollando una Roll para incluir las implementaciones de los protocolos de Myrinet y el gestor de procesos de SCore.

Rocks fue nuestra elección, debido a las siguientes características:

- Facilidad de instalación: sólo es necesario completar la instalación en un nodo, llamado nodo maestro o *frontend*. El resto se instala con Avalache, un programa P2P, de forma completamente automática, evitando tener que instalar los nodos uno a uno.
- Variabilidad de software: es importante que disponga de software que sea fácilmente integrable, evitándonos hacer desarrollos innecesarios.
- Facilidad de mantenimiento. El mantenimiento del cluster se realiza sólo en el nodo maestro (todos los datos de configuración, que conforman el estado, se almacenan en éste).
- Rocks está desarrollado por San Diego Supercomputer Center (SDSC, [11]) que pertenece a National Partnership for Advance Computacional Infrastructure (NPACI [12]). Cuenta con el apoyo de NPACI y tiene un intenso desarrollo, lo que nos garantiza un soporte a largo plazo.
- Está diseñado para minimizar el tráfico por red, es decir, esta preparado para que todos los nodos dispongan de disco duro propio y compartan la información mínima e imprescindible. Esto es muy deseable, pues pensamos utilizarla para cálculo científico intensivo.

Rocks al inicio de este proyecto se encontraba en la 4.3, que es la que hemos utilizado, actualmente ya se encuentra disponible la versión 5.2 "Chimichanga". Rocks 5.2 separa el sistema operativo del middleware permitiendo actualizaciones del sistema operativo (sin desestabilizar el sistema, aislando la base de datos de Rocks de la base de datos del sistema operativo) y mejora el sistema de grafos.

Rocks cubre de forma eficiente la mayor parte de los servicios que necesitamos para administrar y mantener el cluster. Las características de Rocks, que han motivado su elección:

En cuanto a la facilidad de instalación, es uno de los puntos más fuertes de Rocks, en menos de 1 hora somos capaces de realizar una instalación básica de nuestro cluster, siendo perfectamente usable. Y con instalación básica queremos decir, que un usuario podría entrar al sistema, compilar un programa, y ejecutarlo, es decir, que dispondría de las funcionalidades básicas del sistema. Para ello seguimos los pasos indicados en el anexo B.

La variabilidad de software se debe a que el sistema de software se basa en paquetes RPMs, por lo que resulta relativamente fácil encontrar los paquetes en la

red. En caso de necesitarlo, podemos generar los RPM; además Rocks dispone de algunas herramientas de gran utilidad para realizar esta tarea. No obstante, podemos añadir software en la etapa de instalación. Estos módulos de software se llaman Rolls, contienen todo lo necesario para realizar la instalación y la configuración dentro de nuestro sistema de forma automática y además, Rocks tiene casi todo el software que necesitamos en ese formato. Rocks permite decidir desde el *frontend* cómo será la instalación en los nodos esclavos, qué Rolls estarán activas y qué arquitectura se usará.

La facilidad de mantenimiento es, además de lo que ya hemos mencionado, porque incluye un sistema de backup de estado, llamado Roll Restore. Esta Roll trata de guardar los archivos de configuración y scripts, e incluso podemos añadir los archivos que consideremos oportunos. Podemos ver más acerca la configuración y el uso en el anexo E.

Rocks dispone de una base de datos que contiene la información del estado y configuración de los nodos, por lo que con guardar esa base de datos, podremos llevar toda la configuración hardware al mismo punto donde estaba. Pero además es accesible mediante la consola y web, lo que facilitará enormemente las labores de mantenimiento. En la figura 2.7 se muestra la interfaz web a la base de datos implementada vía phpMyAdmin [13] convenientemente adaptada. Las distintas tablas situadas a la izquierda de la figura, son accesibles desde la línea de comando.

El la linea de comandos la figura 2.7 sería:

```
[nacho@ash25 ~]$ rocks list host
HOST           MEMBERSHIP CPUS RACK RANK
ash25:         Frontend   4    0    0
compute-0-6:   Compute    2    0    6
compute-0-5:   Compute    2    0    5
compute-0-0:   Compute    2    0    0
compute-0-1:   Compute    2    0    1
compute-0-2:   Compute    2    0    2
compute-0-3:   Compute    2    0    3
compute-0-4:   Compute    2    0    4
compute-0-12:  Compute    2    0    12
compute-0-8:   Compute    8    0    8
compute-0-9:   Compute    8    0    9
compute-0-10:  Compute    8    0    10
compute-0-11:  Compute    8    0    11
compute-0-13:  Compute    2    0    13
```

## 2.4. Instalación de un cluster con Rocks

Sobre el hardware anteriormente visto, instalamos Rocks siguiendo los pasos indicados en el anexo B, e incluimos varias Rolls:

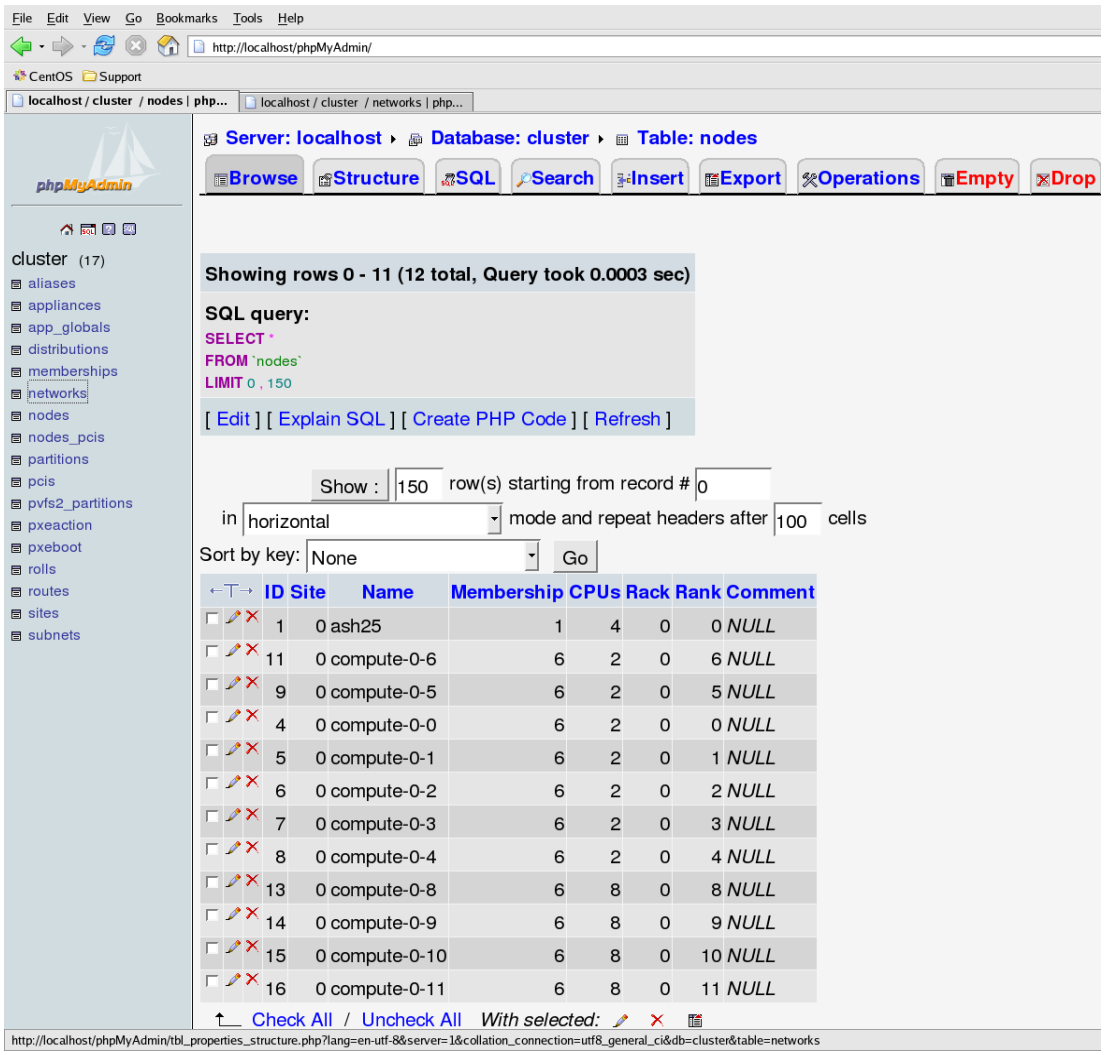



Figura 2.7: Acceso web a la base de datos.

- Rolls Kernel, Base, OS y HPC: son las Rolls básicas del sistema. Base y Kernel nos proveerán de los comandos básicos de Rocks y la instalación por red de los nodos. La Roll OS conforma el sistema operativo básico en este caso el sistema operativo es CentOS 4.3, basado en RedHat Enterprise Level (RHEL) compilado a partir del código fuente. CentOS es el sistema operativo recomendado, podemos cambiarlo por cualquier otro basado en RHEL, cambiando los CD-ROMs o DVDs correspondientes a OS. HPC contiene las implementaciones de MPI (OpenMPI [14], MPICH v1 y MPICH v2 [15]) sobre Ethernet, diversos test de rendimiento, y el servicio 411 (especifico de Rocks, hablaremos posteriormente de él).

- Area51: Esta Roll nos proporciona los servicios básicos de seguridad. Con-

tiene Tripware y Chkrootkit, la primera genera un log de cambios, especificados con anterioridad, con acceso web y correo automático; y la segunda detecta intentos de suplantaciones de identidad.

- Ganglia: Se encarga del servicio de monitorización del cluster, utiliza XML para representar los datos, XDR (*eXternal Data Representation*, definido en el RFC 4506 [16], se sitúa en la capa de presentación) para transportar los datos y RRDTool (*Round Robin Database TOOL* [17]) para almacenar los datos (en una base de datos circular) y generar los gráficos.

- Java: Esta Roll nos resulta interesante para poder manejar la interfaz gráfica de SGE. Contiene la JDK de Sun y otras herramientas.

- SGE [2]: Es fundamental disponer de un sistema de colas, en nuestro caso elegimos este sistema frente a PBS/Torque [18].

- Web-server: Con esta Roll dispondremos del interfaz web (con WordPress) y el wiki del cluster. El servidor web en sí (Apache) ya se incluye en la Roll Base. Será imprescindible para poder monitorizar el cluster desde la web.

- Intel-developer: Nuestro entorno de desarrollo se basa en los compiladores y librerías de Intel ([19], [20] y [21]), por lo que esta Roll es fundamental. En concreto esta Roll contiene la versión 10 de los compiladores de Intel, y ha sido generada por Clustercorp. Dada la importancia que tiene para nosotros, desarrollamos una Roll que incluye los compiladores de Intel en su versión 11.

- Ash25.tsc.uc3m.es-restore: es la Roll Restore realizada sobre el cluster de pruebas. Esta Roll está diseñada para ser un backup de estado del *frontend* (nodo maestro), de forma que contiene archivos de configuración y scripts, no estando orientada a guardar datos de usuario.

Además de todo este software, se fueron añadiendo diversas librerías, tal y como se desarrolla en el próximo capítulo. En él podemos ver los mecanismos de distribución de software. Algunos métodos no son infalibles, y de generar un error, dejan comandos sin ejecutar. Esto nos llevó a desarrollar un pequeño ciclo de pruebas, primero probando este software (no incluido) en un pequeño cluster (un frontend y un nodo esclavo), para implementarlo una vez subsanados los problemas encontrados. Si este software era crítico para la funcionalidad del cluster, se introdujo una etapa inicial de pruebas en un cluster "virtual", esto es emulando la pareja frontend-esclavo.

Como servicios fundamentales en el cluster debemos citar:

- Http: resulta un servicio básico, ya que, forma parte del servicio de instalación de los nodos, pues las peticiones clave se realizan a través de este servicio (petición de fichero de acciones y la petición de paquetes). Y además toda la presentación de la información de monitorización se realizará con http, incluyendo diversos servicios de gestión y mantenimiento.
- Avalanche: es la piedra angular de la instalación en los nodos, está basado en P2P, en concreto en *trackers* BitTorrent. Avalanche distribuye los paquetes de forma escalable.
- SSH (*Secure SHell*): todos los servicios administrativos y de mantenimiento operan sobre este.
- 411: este servicio se encarga de distribuir los ficheros de configuración y contraseñas, entre el maestro y los esclavos, mantiene la coherencia de configuración para los usuarios, y otros servicios. El servicio 411 notifica a los nodos, desde el maestro, de la existencia de cambios en los ficheros (típicamente, ficheros de configuración), éstos piden el envío de los ficheros al maestro, y el maestro los sirve por http codificados. Esta basado en NIS (*Network Information Service* [22]), un extendido sistema para distribuir contraseñas en UNIX), utiliza criptografía asimétrica (RSA) de 1024 bits de longitud por defecto.
- DNS: el servidor de nombres, opera a través de la base de datos, MySQL, del cluster, se actualiza al añadir nodos o por comandos.
- Correo: disponemos de un servidor Postfix [23] que se encargará de enviar el correo, tras una mínima configuración por parte del usuario.
- SGE: resulta imprescindible. En un cluster, diseñado para ejecutar trabajos de forma intensiva, necesitamos un servicio de colas, y gestión de estas colas, para poder aprovechar al máximo toda la potencia de cálculo disponible.

Tenemos que destacar, aunque no sea fundamental, que Rocks sigue el concepto SSI (*Single System Image*) desde el punto de vista de los usuarios, orientando el cluster a que vean un sistema único, con un sólo sistema de ficheros (exportando el home de cada usuario a los nodos). De este modo, la centralización de los servicios en el nodo maestro o *frontend*, facilita la administración

### 2.4.1. Web del cluster

La necesidad de una Web viene motivada por ofrecer una interfaz amigable para los usuarios, donde puedan encontrar información útil acerca del estado de ejecución de sus programas, guías de uso y ayuda. Si añadimos que puede ser usado

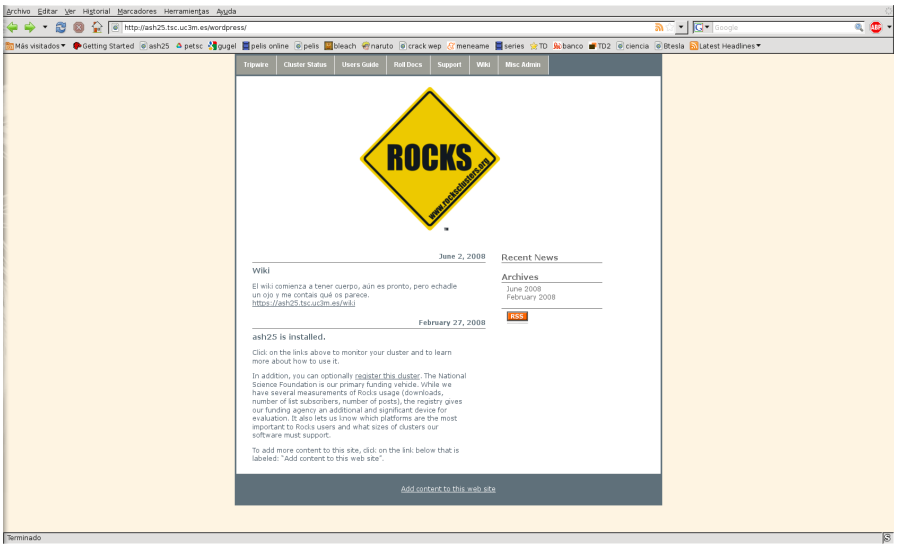



Figura 2.8: Web principal de ash25.

para las tareas de monitorización, gestión y administración, tenemos un servicio muy útil.

En la figura 2.8 podemos ver la página principal del cluster[1].

Desde esta página podemos acceder a:

- monitorizar el cluster, en la solapa Ganglia
- a diversas guías de usuario, en las dos siguientes solapas
- en Misc Admin podemos ver el grafo de la distribución, administrar la base de datos, y crear las etiquetas para nuestras máquinas con sus direcciones IP y MAC.
- la última solapa es Wiki, sin lugar a dudas, la mejor forma de ofrecer guías y ayudas, debido a que es una de las formas más rápidas de compartir la información.

Podremos ver la Web del Wiki[2] en la figura 2.9.

En el wiki se introdujo información sobre instalación y gestión del cluster, además de incluir una gran variedad de guías que ayudarán a dar los primeros pasos en este entorno.

[1] https://ash25.tsc.uc3m.es/ o http://ash25.tsc.uc3m.es/
[2] https://ash25.tsc.uc3m.es/wiki/ o http://ash25.tsc.uc3m.es/wiki/

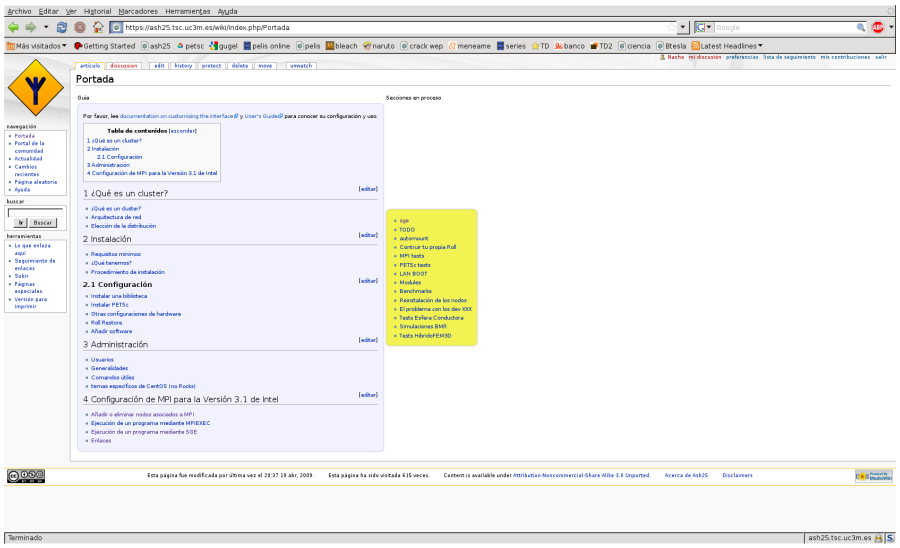



Figura 2.9: Web del wiki.

### 2.4.2. Monitor de recursos

La monitorización con Ganglia es simple una vez instalado, viene configurada una Web[3] donde se muestran los gráficos que detallan el estado del cluster. Podemos ver la página principal de la Web en la figura 2.10.

Y como podemos ver en la figura 2.8 también podremos acceder desde la página principal. Además cuenta con un monitor de la cola de trabajo, donde podremos ver el estado de nuestros trabajos sin tener que entrar en el equipo. Los monitores nos dan de forma individualizada (por cada equipo) y global (del cluster) la ocupación de CPU, memoria RAM, de la red, disco duro, etc.

### 2.4.3. Gestor de colas

Como hemos dicho anteriormente, la disponibilidad de un gestor de colas es fundamental con el crecimiento del número de usuarios, ya que permite el máximo aprovechamiento de los equipos. En nuestro caso el gestor de colas elegido es SGE. Podríamos haber elegido PBS/Torque que se encuentra disponible, pero SGE demostró poseer características que satisfacen todas nuestras necesidades, además de ser ampliamente aceptado en la comunidad y recomendado por Rocks.

[3]https:/ash25.tsc.uc3m.es/ganglia o http:/ash25.tsc.uc3m.es/ganglia


Figura 2.10: Monitorización con Ganglia.

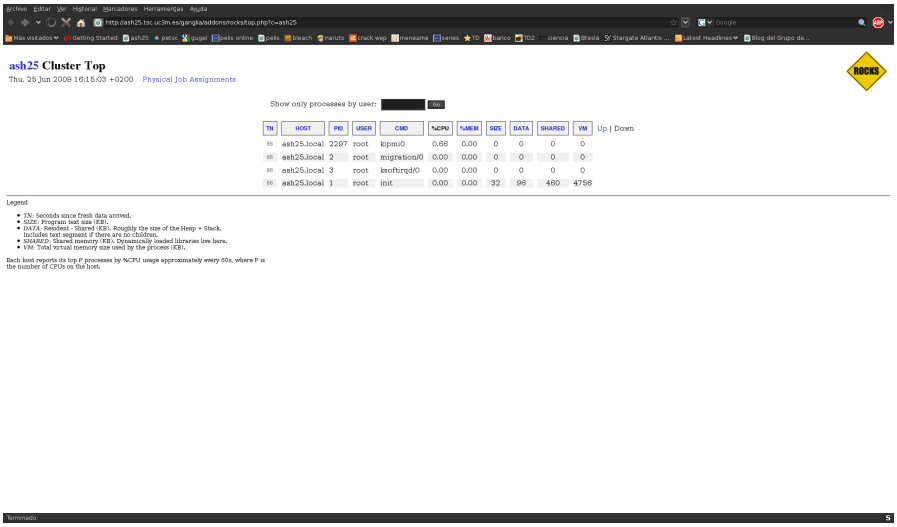



Figura 2.11: Lista de procesos activos en el cluster desde Ganglia.

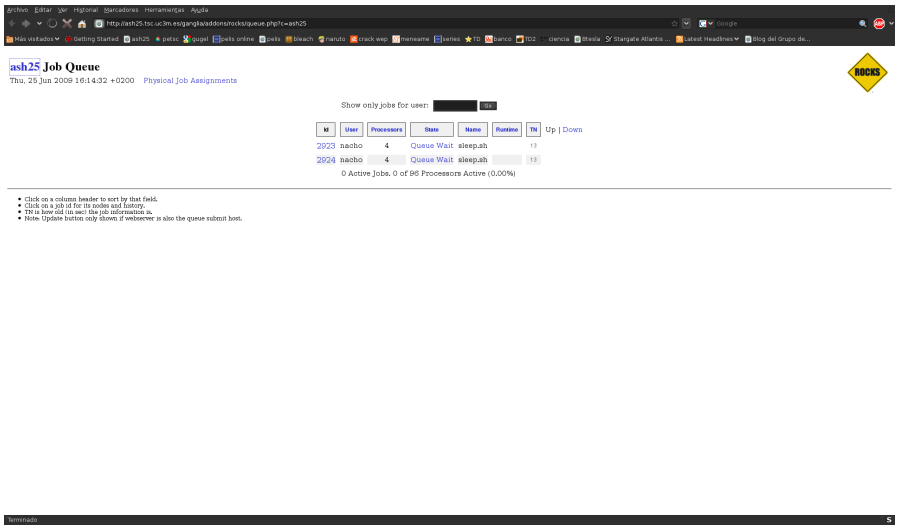



Figura 2.12: Visor de trabajos con Ganglia.

Sun Grid Engine (SGE), desarrollado por Sun Microsystems y conocido anteriormente como CODINE (COmputing in DIstributed Networked Environments) o GRD (Global Resource Director), es un sistema de colas de código abierto. Podemos encontrar más información en el anexo C, en [2] o en [24].

Este software se encarga de aceptar, planificar y enviar las tareas remotamente a los nodos que componen el cluster. También se encarga de planificar y otorgar los recursos (distribuidos), tales como procesadores, memoria, espacio en disco y licencias de software.

Soporta nodos heterogéneos, por lo que resulta muy interesante. Además cuenta con un monitor que nos permite ver el estado de todas las colas, mandar trabajos y modificar cualquier parámetro. Este monitor se invoca directamente en la consola con el comando *qmon*.

Podemos ver más acerca del uso y manejo de SGE en el anexo C

### 2.4.4. Entorno de ejecución

Por entorno de ejecución entendemos qué variables de entorno están disponibles y cómo podemos hacer uso de las mismas. Este punto es de gran importancia pues no deberemos mezclar entornos. Podemos comprobarlo en la siguiente sección, donde tendremos que distinguir entre compiladores y librerías de paso de mensajes.

En el cluster se encuentran disponibles compiladores de GNU [25] y de Intel (en dos versiones) [19], además de varias implementaciones de MPI (MPICH [15], OpenMPI [14] e Intel MPI [21]), librerías de calculo como MKL (*Math Kernel Library* [20]), METIS ([26] un conjunto de programas para el particionado de mallas, que produce reordenamientos para matrices dispersas) y MUMPS (*MUltifrontal Massively Parallel Solver*, [27]). Algunas de ellas son incompatibles con otras, por lo que necesitamos un control exhaustivo del entorno. Para llevar a cabo esta tarea se empleó *modules*, por lo que añadir una librería al entorno se reduce a un comando.

La librería *modules* controla dinámicamente el entorno de ejecución, permitiendo tanto el añadir variables y librerías, como eliminarlas. La clave radica en generar un script en Tcl para cada librería, con los comandos aplicamos los scripts modificando las variables de entorno. Podemos ver más detalles en el anexo D y en [3].

## 2.5. MPI

Debemos aclarar que la programación en paralelo se plantea de forma distinta a como procederíamos para un programa secuencial, debemos considerar qué recursos son compartidos y cuáles no lo son, y adecuarnos a estos recursos.

En una programación secuencial, típicamente, disponemos de una CPU (un core), un conjunto de instrucciones y memoria, para resolver un problema, sin embargo, ahora disponemos de varias CPU (con varios cores a su vez), instrucciones especificas para cada core (proceso) y distintas memorias y canales (caché, RAM, Ethernet,...), para resolver el mismo problema. Por lo que deberemos planificar en la medida de lo posible, los recursos y los intercambios de datos.

Debido al modelo actual de arquitectura, la memoria es compartida y distribuida, cada procesador comparte la memoria cache entre todos sus cores y la RAM entre los procesadores, y distribuye para el resto de procesadores conectados a la red. Para solucionar estos problemas anteriormente existían un lenguaje para cada arquitectura por lo que migrar el código era impensable, hoy en día gracias a MPI disponemos de un interfaz definido, que mitigará los problemas de concurrencia, coherencia, de una manera eficiente y para lenguajes conocidos, como C o FORTRAN.

Lo que hace tan especial a MPI es que está basado en canales de comunicación y no en sockets, es decir, si tenemos $N$ equipos interconectados por sockets, tendríamos $(N-1)N/2$ conexiones punto a punto. Mientras que por canales

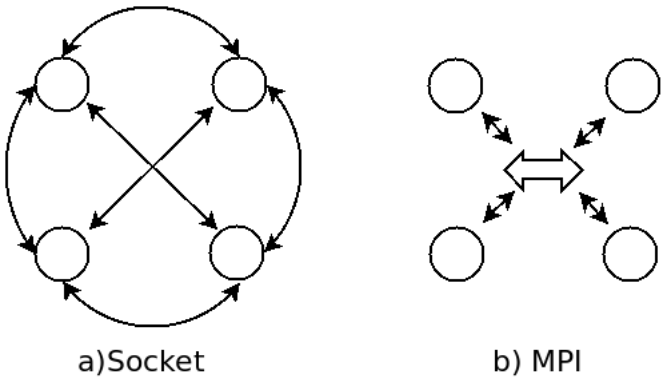



Figura 2.13: Comparación entre sockets y MPI.

virtuales tendremos $N$ conexiones al canal. Estos canales virtuales, pueden o no implementarse por sockets, pero conservan la idea de un canal para los nodos y no una conexión para cada par de nodos. En el ejemplo de la figura 2.13, vemos como para cuatro nodos por sockets hacen falta seis conexiones, mientras que para MPI hacen falta cuatro. Esta filosofía mejora el retardo y la capacidad de la red, además de poseer una mejor escalabilidad.

Gracias a esta capa podremos tener procesos concurrentes cooperantes, de forma sencilla.

Dado que es un estándar, comprobaremos varias implementaciones del mismo. En concreto, a continuación ilustraremos como trabajar con MPICH, en dos variantes (usando compiladores GNU y usando los compiladores de Intel), IntelMPI y OpenMPICH. Todas las implementaciones citadas corresponden al estándar MPI v1.0, salvo IntelMPI que implementa MPI-v2.0). Veamos a continuación algunos programas de ejemplo diversas compilaciones y lincados así como su ejecución.

### 2.5.1. Configurando el entorno de trabajo: compilando y ejecutando en paralelo.

Para ejecutar un programa en paralelo necesitaremos una lista de ordenadores, para generarlos hacemos uso de los comandos de Rocks, que nos dan acceso a la base de datos de nodos, directamente en el terminal:

- OpenMPI:

```
>rocks list host|awk '/compute/{system("ssh " substr($1,0,length($1)-1)
    " echo " substr($1,0,length($1)-1))}'|awk '/^compute/{system("rocks
     list host "$1)}'|awk '/^Compute/{print "compute-"$3"-"$4" slots="
    $2}'>machines.conf
>cat machines.conf
ash25 slots=4
compute-0-5 slots=2
compute-0-0 slots=2
compute-0-1 slots=2
compute-0-2 slots=2
compute-0-3 slots=2
compute-0-4 slots=2
compute-0-6 slots=2
compute-0-8 slots=8
compute-0-10 slots=8
compute-0-9 slots=8
compute-0-11 slots=8
```

- MPICH:

```
rocks list host|awk '/compute/{system("ssh " substr($1,0,length($1)-1)"
     echo " substr($1,0,length($1)-1))}'|awk '/^compute/{system("rocks
    list host "$1)}'|awk '/^Compute/{print "compute-"$3"-"$4":"$2}'>
    machines.conf
cat machines.conf
ash25:4
compute-0-0:2
compute-0-1:2
compute-0-2:2
compute-0-4:2
compute-0-5:2
compute-0-6:2
compute-0-8:8
compute-0-10:8
compute-0-9:8
compute-0-11:8
```

- Intel MPI:

  Para Intel MPI la lista de nodos es igual a la que obtenemos para MPICH, pero además necesitaremos un fichero llamado `.mpd.conf` con permisos de solo lectura para el usuario, que contenga algo similar a lo siguiente:

```
MPD_SECRETWORD=my_secret_word
```

La diferencia principal entre los tres implementaciones radica en que OpenMPI necesita saber cuantos cores por nodo tenemos, mientras que el resto de implementaciones no lo necesita, es decir, para OpenMPI si necesitamos poner `slots=N` con N el número de cores totales de la máquina, mientras que para el resto no es indispensable poner `:N`.

### 2.5.2. Programando en C

Para ejemplificar este punto usaremos el siguiente programa escrito en el fichero 2.1.

**Fichero 2.1: mpi_test.c**

```
/*INCLUDE SECTION*/
#include <stdio.h>
#include <mpi.h>
/*MAIN PROGRAM*/
int main(int argc, char **argv) {
/*VARIABLE SECTION*/
char* name=(char*)calloc(MPI_MAX_PROCESSOR_NAME, sizeof(char));
    int resultlen;
    int ierror, rank, size;
/*CODE SECTION*/
    MPI_Init(&argc,&argv);//esto inicializa MPI
    ierror=MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    if(ierror)
        printf("error en MPI_Comm_rank()\n");
    ierror=MPI_Comm_size(MPI_COMM_WORLD,&size);
    if(ierror)
        printf("error en MPI_Comm_size()\n");
    ierror= MPI_Get_processor_name(name, &resultlen);
    int i=0;double j=1;
    for(;i<100000000;i++){
        j=i/j;
    }
printf("Hello world! I am %d out of %d in %s.\n",rank,size,name);
    MPI_Finalize();
    free(name);
    return 0;
}
```

El código en este caso es un programa muy básico, diseñado con el fin de ocupar el core[4] del procesador al máximo, lo que nos servirá para asegurarnos de su correcto funcionamiento.

Podemos ver como en la cabecera del fichero al que llamaremos `mpi_test.c` se incluyen las definiciones de MPI (en el fichero `mpi.h`), y además en el código se incluyen las funciones `MPI_Init()` y `MPI_Finalize()` que sirven para inicializar y finalizar, respectivamente, el entorno paralelo.

Una vez inicializado el entorno preguntamos a la implementación de MPI en qué proceso estamos (numerado del 0 al N-1) y en cuantos procesos estamos ejecutando el programa. El número de procesos será el mismo para cada uno de los procesos pero el procesador en el que estamos variará.

[4]Los programas diseñados para alto rendimiento hacen uso de procesos (pesados), no de hilos, por lo que existe la correspondencia entre cores y procesos, debido a que sólo es posible un proceso por core como máximo.

### OpenMPI con los compiladores de GNU

Configuramos las variables de entorno y comprobamos que efectivamente disponemos de la versión correcta.

```
>module clear
>module add openmpi
>mpicc --showme
gcc -I/usr/include/openmpi -I/usr/include/openmpi/openmpi -m64 -pthread -L/
    usr/lib64/openmpi -lmpi -lorte -lopal -ldl -Wl,--export-dynamic -lnsl -
    lutil -lm -ld
```

Procederemos a compilar como sigue:

```
mpicc -o mpi_test mpi_test.c
```

Y con el fichero de nodos activos, ejecutamos para 16 procesos:

```
mpirun -np 16 -machinefile ~/machines.conf mpi_test
```

Obteniendo como resultado:

```
Hello world! I am 9 out of 16 in compute-0-8.local.
Hello world! I am 6 out of 16 in compute-0-9.local.
Hello world! I am 14 out of 16 in compute-0-8.local.
Hello world! I am 13 out of 16 in compute-0-9.local.
Hello world! I am 8 out of 16 in compute-0-8.local.
Hello world! I am 15 out of 16 in compute-0-8.local.
Hello world! I am 3 out of 16 in compute-0-8.local.
Hello world! I am 12 out of 16 in compute-0-8.local.
Hello world! I am 7 out of 16 in compute-0-9.local.
Hello world! I am 0 out of 16 in compute-0-9.local.
Hello world! I am 1 out of 16 in compute-0-8.local.
Hello world! I am 2 out of 16 in compute-0-9.local.
Hello world! I am 4 out of 16 in compute-0-9.local.
Hello world! I am 5 out of 16 in compute-0-8.local.
Hello world! I am 11 out of 16 in compute-0-9.local.
Hello world! I am 10 out of 16 in compute-0-9.local.
```

En un tiempo de ejecución de:

```
real    0m5.882s
user    0m0.027s
sys     0m0.029s
```

### MPICH con los compiladores de GNU

Otra vez, limpiamos el entorno, comprobamos y compilamos con la implementación de MPICH y los compiladores de GNU:

```
>module clear
>module add mpich_gnu
>mpicc -o mpi_test mpi_test.c -v
mpicc for 1.2.7 (release) of : 2005/06/22 16:33:49
Reading specs from /usr/lib/gcc/x86_64-redhat-linux/3.4.6/specs
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir
    =/usr/share/info --enable-shared --enable-threads=posix --disable-
    checking --with-system-zlib --enable __cxa_atexit --disable-libunwind-
    exceptions --enable-java-awt=gtk --host=x86_64-redhat-linux
       [...]
>mpicc -o mpi_test mpi_test.c
```

Ejecutando de la misma manera, pero cuidando de que el fichero de configuración de los nodos activos sea el correcto:

```
mpirun -np 16 -machinefile ~/machines.conf mpi_test
```

El resultado obtenido es similar:

```
Hello world! I am 11 out of 16 in compute-0-9.local.
Hello world! I am 13 out of 16 in compute-0-8.local.
Hello world! I am 8 out of 16 in compute-0-8.local.
Hello world! I am 15 out of 16 in compute-0-8.local.
Hello world! I am 7 out of 16 in compute-0-8.local.
Hello world! I am 0 out of 16 in compute-0-9.local.
Hello world! I am 9 out of 16 in compute-0-8.local.
Hello world! I am 6 out of 16 in compute-0-8.local.
Hello world! I am 14 out of 16 in compute-0-9.local.
Hello world! I am 1 out of 16 in compute-0-9.local.
Hello world! I am 3 out of 16 in compute-0-8.local.
Hello world! I am 12 out of 16 in compute-0-9.local.
Hello world! I am 2 out of 16 in compute-0-8.local.
Hello world! I am 4 out of 16 in compute-0-9.local.
Hello world! I am 5 out of 16 in compute-0-9.local.
Hello world! I am 10 out of 16 in compute-0-9.local.
```

Pero el tiempo de ejecución se incrementa:

```
real    0m7.888s
user    0m1.024s
sys     0m1.897s
```

### MPICH con los compiladores de Intel

La compilación es idéntica a la anterior, pero con los compiladores de Intel:

```
>module clear
>module add mpich_intel
>mpicc -o mpi_test mpi_test.c -v
mpicc for 1.2.7 (release) of : 2005/11/04 11:54:51
Version 10.1
/opt/intel/cce/10.1.011/bin/mcpcom    --g -mP3OPT_inline_alloca -D__HONOR_STD
     -D__ICC=1010 -D__INTEL_COMPILER=1010 -D_MT "--Asystem(unix)" -D__ELF__
    "--Acpu(x86_64)" "--Amachine(x86_64)" -D__INTEL_COMPILER_BUILD_DATE
    =20071116 -D__PTRDIFF_TYPE__=long "-D__SIZE_TYPE__=unsigned long" -
    D__WCHAR_TYPE__=int "-D__WINT_TYPE__=unsigned int" "-D__INTMAX_TYPE__=
    long int" "-D__UINTMAX_TYPE__=long unsigned int" -D__QMSPP_ -
    D__OPTIMIZE__
```

```
        [...]
>mpicc -o mpi_test mpi_test.c
```

La ejecución es análoga a la anterior, y aunque el fichero de nodos activos del caso anterior sigue siendo válido, es recomendable generarlo antes de cada uso.

```
mpirun -np 16 -machinefile ~/machines.conf mpi_test
```

Otra vez la salida por pantalla es similar, debido a la concurrencia es difícil que sea completamente igual.

```
Hello world! I am 0 out of 16 in compute-0-8.local.
Hello world! I am 13 out of 16 in compute-0-9.local.
Hello world! I am 5 out of 16 in compute-0-9.local.
Hello world! I am 1 out of 16 in compute-0-9.local.
Hello world! I am 9 out of 16 in compute-0-9.local.
Hello world! I am 7 out of 16 in compute-0-9.local.
Hello world! I am 3 out of 16 in compute-0-9.local.
Hello world! I am 11 out of 16 in compute-0-9.local.
Hello world! I am 15 out of 16 in compute-0-9.local.
Hello world! I am 8 out of 16 in compute-0-8.local.
Hello world! I am 4 out of 16 in compute-0-8.local.
Hello world! I am 12 out of 16 in compute-0-8.local.
Hello world! I am 10 out of 16 in compute-0-8.local.
Hello world! I am 2 out of 16 in compute-0-8.local.
Hello world! I am 6 out of 16 in compute-0-8.local.
Hello world! I am 14 out of 16 in compute-0-8.local.
```

Otra vez el tiempo de ejecución es muy elevado:

```
real    0m7.665s
user    0m0.176s
sys     0m0.384s
```

### Intel MPI con los compiladores de Intel

Una vez configurados los ficheros `.mpd.conf` y `machines.conf`, limpiamos el entorno y cargamos la librería. Compilamos de la misma manera usando `mpiicc`, en lugar de `mpicc`:

```
>module clear
>module add impi
>mpiicc -v
mpiicc for the Intel(R) MPI Library 3.1 for Linux
Version 10.1
        [...]
>mpiicc -o mpi_test mpi_test.c
```

Para ejecutar, primero hemos de crear los sockets de comunicación, con `mpdboot`, y posteriormente ejecutar con `mpiexec`:

```
mpdboot -n 7 -f machines.conf -r ssh
mpiexec -l -n 16 -path $PWD mpi_test
```

**Fichero 2.2: mpi_test.F90**

```
      program main
2     include 'mpif.h'
      character(LEN=20) :: name
      integer :: resultlen, ierror, rank, size,i
      REAL(kind=8) :: j
      call MPI_Init(ierror)
7     call MPI_Comm_rank(MPI_COMM_WORLD,rank,ierror)
      call MPI_Comm_size(MPI_COMM_WORLD,size,ierror)
      call MPI_Get_processor_name(name, resultlen,ierror)
      j = 1
      do i = 0, 100000000, 1
12        j = i / j
      enddo
      print *,"Hello world! I am ",rank," out of ",size," in ",name,"."
      call MPI_Finalize(ierror)
      end program main
```

El resultado que obtenemos es:

```
Hello world! I am 9 out of 16 in compute-0-8.local.
Hello world! I am 6 out of 16 in compute-0-8.local.
Hello world! I am 14 out of 16 in compute-0-8.local.
Hello world! I am 13 out of 16 in compute-0-8.local.
Hello world! I am 8 out of 16 in compute-0-9.local.
Hello world! I am 15 out of 16 in compute-0-8.local.
Hello world! I am 3 out of 16 in compute-0-9.local.
Hello world! I am 12 out of 16 in compute-0-9.local.
Hello world! I am 7 out of 16 in compute-0-9.local.
Hello world! I am 0 out of 16 in compute-0-8.local.
Hello world! I am 1 out of 16 in compute-0-8.local.
Hello world! I am 2 out of 16 in compute-0-8.local.
Hello world! I am 4 out of 16 in compute-0-9.local.
Hello world! I am 5 out of 16 in compute-0-9.local.
Hello world! I am 11 out of 16 in compute-0-9.local.
Hello world! I am 10 out of 16 in compute-0-9.local.
```

Pero en un tiempo sorprendente:

```
real    0m0.551s
user    0m0.089s
sys     0m0.021s
```

### 2.5.3. Programando en Fortran 90

Portamos el código del ejemplo anterior en C a Fortran 90, obteniendo el fichero 2.2.

Los cambios en el fichero, que se denomina ahora *mpi_test.F90*, son muy ligeros: el fichero que debemos incluir se llama ahora *mpif.h* y hemos evitado la reserva de memoria dinámica por simplicidad. Por lo demás, el fichero *mpi_test.F90* es idéntico a su homólogo en C.

### OpenMPI con los compiladores de GNU

Siguiendo el esquema anterior, compilamos:

```
>module clear
>module add openmpi
>mpif90 -o help  -showme
gfortran -I/usr/include/openmpi -I/usr/include/openmpi/64 -m64 -pthread -I/
    usr/lib64/openmpi -o help -L/usr/lib64/openmpi -lmpi_f90 -lmpi -lorte -
    lopal -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl
>mpif90 -o mpi_test mpi_test.F90
```

La ejecución es exactamente igual que cuando lo realizábamos en C:

```
mpirun -np 16 -machinefile ~/machines.conf mpi_test
```

Y obtenemos:

```
Hello world! I am      2  out of   16  in compute-0-8.local   .
Hello world! I am      3  out of   16  in compute-0-8.local   .
Hello world! I am      0  out of   16  in compute-0-8.local   .
Hello world! I am      1  out of   16  in compute-0-8.local   .
Hello world! I am      5  out of   16  in compute-0-8.local   .
Hello world! I am      6  out of   16  in compute-0-9.local   .
Hello world! I am      9  out of   16  in compute-0-9.local   .
Hello world! I am     12  out of   16  in compute-0-9.local   .
Hello world! I am     13  out of   16  in compute-0-9.local   .
Hello world! I am     10  out of   16  in compute-0-9.local   .
Hello world! I am     11  out of   16  in compute-0-9.local   .
Hello world! I am     14  out of   16  in compute-0-9.local   .
Hello world! I am     15  out of   16  in compute-0-8.local   .
Hello world! I am      8  out of   16  in compute-0-8.local   .
Hello world! I am      7  out of   16  in compute-0-9.local   .
Hello world! I am      4  out of   16  in compute-0-8.local   .
```

Con un tiempo de:

```
real    0m3.715s
user    0m0.069s
sys     0m0.038s
```

### MPICH con los compiladores de GNU

Se puede compilar programas en Fortran 77, pero no en Fortran 90. Como el presente proyecto se orienta al uso de Fortran 90, no concluimos este test.

### MPICH con los compiladores de Intel

Compilamos:

```
>module clear
>module add mpich_intel
>mpif90  -o mpi_test mpi_test.c -v
mpif90 for 1.2.7 (release) of : 2005/11/04 11:54:51
Version 10.1
/opt/intel/fce/10.1.011/bin/fortcom     -mP1OPT_version=1010 -
        [...]
>mpif90 -o mpi_test mpi_test.F90
```

La ejecución es exactamente igual que cuando lo realizabamos en C:

```
mpirun -np 16 -machinefile ~/machines.conf mpi_test
```

Los resultados son:

```
Hello world! I am      0  out of   16  in compute-0-8.local   .
Hello world! I am      7  out of   16  in compute-0-8.local   .
Hello world! I am     14  out of   16  in compute-0-8.local   .
Hello world! I am      2  out of   16  in compute-0-8.local   .
Hello world! I am      9  out of   16  in compute-0-8.local   .
Hello world! I am     13  out of   16  in compute-0-8.local   .
Hello world! I am      1  out of   16  in compute-0-8.local   .
Hello world! I am     12  out of   16  in compute-0-8.local   .
Hello world! I am     11  out of   16  in compute-0-9.local   .
Hello world! I am      6  out of   16  in compute-0-9.local   .
Hello world! I am      5  out of   16  in compute-0-9.local   .
Hello world! I am      3  out of   16  in compute-0-9.local   .
Hello world! I am      4  out of   16  in compute-0-9.local   .
Hello world! I am      8  out of   16  in compute-0-9.local   .
Hello world! I am     10  out of   16  in compute-0-9.local   .
Hello world! I am     15  out of   16  in compute-0-9.local   .
```

Otra vez en un tiempo elevado:

```
real    0m7.965s
user    0m0.185s
sys     0m0.319s
```

### Intel MPI con los compiladores de Intel

Como en la versión en C, debemos compilar usando un comando diferente:

```
mpifort -o mpi_test mpi_test.F90
```

La ejecución es exactamente igual que cuando lo realizábamos en C:

```
mpdboot -n 7 -f machines.conf -r ssh
mpiexec -l -n 16 -path $PWD mpi_test
```

Obteniendo, curiosamente:

```
Hello world! I am     0  out of    16  in compute-0-9   .
Hello world! I am     1  out of    16  in compute-0-9   .
Hello world! I am     2  out of    16  in compute-0-9   .
Hello world! I am     3  out of    16  in compute-0-9   .
Hello world! I am     5  out of    16  in compute-0-9   .
Hello world! I am     4  out of    16  in compute-0-9   .
Hello world! I am     9  out of    16  in compute-0-9   .
Hello world! I am     6  out of    16  in compute-0-9   .
Hello world! I am     7  out of    16  in compute-0-8   .
Hello world! I am     8  out of    16  in compute-0-8   .
Hello world! I am    10  out of    16  in compute-0-8   .
Hello world! I am    11  out of    16  in compute-0-8   .
Hello world! I am    13  out of    16  in compute-0-8   .
Hello world! I am    12  out of    16  in compute-0-8   .
Hello world! I am    14  out of    16  in compute-0-8   .
Hello world! I am    15  out of    16  in compute-0-8   .
```

Otra vez, con un tiempo record:

```
real    0m0.606s
user    0m0.099s
sys     0m0.013s
```

## 2.6. Conclusiones

El cluster que necesitamos es de tipo HPC, y necesitamos que sea fácil de administrar, que nos ayude con la congruencia de software, así como con el control de las tareas paralelas.

Rocks ha resultado ser la distribución que posee todos los elementos deseables, así como infinidad de herramientas que facilitarán nuestra labor.

Las primeras pruebas realizadas con MPI demuestran que Intel MPI será la herramienta más eficiente, y por lo tanto la empleada desde este punto.

# Capítulo 3

# Instalación y configuración del software en el cluster

En el presente capitulo nos centraremos en explicar cómo es el proceso de añadir software en Rocks. Como hemos mencionado anteriormente, Rocks está basado en paquetes RPMs (*Redhat Package Manager*) , y desde el *frontend* o nodo maestro tenemos la posibilidad de decidir cómo será la instalación en el resto de nodos.

Podemos añadir el software de diversas maneras, nuestras opciones serán distintas dependiendo del estado del cluster y del tipo de librería.

Por poner un ejemplo práctico: *modules*, la librería que controla las variables de entorno, fue instalada solamente en el *frontend*, pues no tendremos la necesidad de situarla en los nodos (salvo muy raras excepciones, no es necesario hacer "login" en los nodos). En el anexo D podemos encontrar más información sobre esta librería

Por lo tanto el objetivo práctico de este capítulo será presentar las diversas formas de añadir software, mediante ejemplos prácticos:

- El fichero de configuración `rc.modules`: fue necesario crear y modificar este fichero de configuración. En nuestro caso carga los módulos IPMI[1], para monitorización del cluster. Al ser necesario en todos los nodos, se distribuyó por el servicio 411.

[1] *Intelligent Platform Management Interface*, es un estándar de monitorización y control de equipos, podemos leer más en http://www.intel.com/design/servers/ipmi/spec.htm.

- IPMItool: este software para monitorización y administración remoto fue añadido vía RPM. Podemos hacerlo de dos formas:
  - Descargando el RPM de algún repositorio.
  - Creando el RPM en nuestra máquina, típicamente a partir de las fuentes. Lo haremos de dos formas, con las herramientas de Rocks y sin ellas.
- MUMPS: esta librería matemática contiene un resolvedor multifrontal, que utilizaremos en posteriores capítulos. Veremos dos formas de instalarlo:
  - Instalación vía NFS: esta es la forma más rápida y menos eficiente. La idea radica en usar el servicio NFS del cluster para distribuir la librería en tiempo de ejecución (carga de la librería). NFS la distribuirá siempre que sea necesaria, este método incrementa notoriamente el tráfico de red y es diametralmente opuesto a nuestra filosofía, pero es el método más rápido.
  - Instalación local: la librería se instala localmente (sistema de archivos local sobre disco duro de cada nodo de cálculo). La instalación local se controla desde el nodo maestro. Sólo se usa NFS para distribuir la librería en el proceso de instalación inicial.
- Intel Compiler Suite y MKL: forman parte de nuestro entorno básico de desarrollo, por lo que el método empleado será la Roll.

De todos los métodos, la creación de la Roll es el más estable, duradero y recomendado. En caso de tener la necesidad de reinstalar el cluster, será el método más rápido y efectivo.

## 3.1. Distribuir archivos de configuración con el servicio 411

En nuestro caso fue necesario retocar el fichero `/etc/rc.modules`, para incluir los módulos de IMPItool en el kernel del sistema. Además debe ser igual en todos y cada uno de los nodos y que en caso de modificar el fichero en el *frontend*, el cambio se debe reproducir en los nodos, por lo que, en este caso el servicio 411 es el más indicado. El proceso es simple, debemos añadir el fichero al listado y notificar los cambios. Al notificar los cambios el servicio distribuirá automáticamente el fichero.

Una vez que tenemos el fichero `/etc/rc.modules`, modificamos el fichero `/var/411/FILES.mk`:

**Fichero 3.1: /var/411/FILES.mk**

```
FILES += /etc/rc.modules
```

Y notificamos los cambios:

```
make -C /var/411 -force
```

Y habremos completado la tarea, en caso de variar el fichero y no notificar los cambios, estos tomarán cuerpo en cinco horas. Esto es debido a que esta sincronización esta programada en el cron.

## 3.2. Instalación vía paquete RPM

Posiblemente es uno de los métodos más simples. Nos servirá para introducir los elementos básicos de Rocks.

Disponemos de tres métodos para conseguir un paquete RPM:

- Construirlo con las herramientas proporcionadas por el paquete, de forma ajena a Rocks. También podríamos crearlo con `rpmbuild` u otras herramientas.
- Crearlo haciendo uso de las herramientas que nos proporciona Rocks.
- Descargarlo de un repositorio, cuidando de la arquitectura y compatibilidades.

Estos métodos están listados de mayor a menor dificultad, por lo que comenzaremos en este orden, partiendo del código fuente, ya que una vez construido el rpm la instalación es la misma para los tres métodos.

Así pues descargamos el código fuente; en nuestro caso:

```
wget http://nfsi.dl.sourceforge.net/sourceforge/ipmitool/ipmitool-1.8.11.tar.
    gz
```

Descomprimimos y nos colocamos en el directorio que se crea, tecleando en el terminal:

```
tar -xzf ipmitool-1.8.11.tar.gz
cd ipmitool-1.8.11
```

Y procedemos a configurar y construir el RPM:

```
./configure
make rpm
```

Tendremos el RPM en el subdirectorio `rpmbuild/RPMS` y estará configurado para nuestra arquitectura.

Ahora procederemos con las herramientas de Rocks; en este caso usaremos una aproximación a la creación parcial de una Roll.

Una vez descargado el paquete `ipmitool-1.8.11.tar.gz`:

```
cd /state/partition1/site-roll/rocks/src/roll
./bin/make-roll-dir.py -n ipmitool -v 1.8.11
cd ipmptool
```

Debemos tener cuidado dándole el mismo nombre (`-n`) y versión (`-v`) que tiene el paquete en cuestión. A continuación retocamos el fichero `version.mk` del directorio `src/ipmitool`, para hacer coincidir la extensión del paquete:

**Fichero 3.2: Makefile src/ipmitool/version.mk**

```
NAME            = ipmitool
VERSION         = 1.8.11
RELEASE         = 1
TARBALL_POSTFIX = tar.gz
```

Copiamos el paquete en ese mismo directorio y procedemos a realizar el RPM:

```
cp ~/ipmitool-1.8.11.tar.gz src/ipmitool
make rpms
```

En unos minutos encontraremos el paquete RPM, `ipmitool-1.8.11-1.x86_64.rpm` en el directorio `RPMS/x86_64`.

Este es el punto en el que en las instalaciones dispondremos del paquete RPM.

### 3.2.1. Instalación del paquete RPM en el *frontend*

Para completar la instalación en el *frontend* simplemente tenemos que introducir:

```
rpm -i ipmitool-1.8.11-1.ia32e.rpm
```

Esta instalación no perdura ante fallos críticos, es decir, que si por algún motivo necesitamos reinstalar el frontend, debemos volver a instalar este paquete. Como veremos posteriormente, podrémos añadir este paquete a una Roll, y de esta forma instalarlo automáticamente al reinstalar (o al realizar una nueva instalación, en caso de que queramos instalar un segundo *frontend*).

### 3.2.2. Instalación del paquete RPM en los nodos

Ahora nos dirigiremos al directorio `/home/install/site-profiles/4.3/nodes/` y generamos un fichero XML (en caso de no existir), llamado `extend-compute.xml`, partiendo de la plantilla existente:

```
cd /home/install/site-profiles/4.3/nodes/
cp skeleton.xml extend-compute.xml
```

Ahora añadimos el paquete donde nos propone la plantilla en `extend-compute.xml`:

**Fichero 3.3: extend-compute.xml de IPMItool**

```
<package>ipmitool</package>
```

Cargamos el RPM en el directorio de la distribución y reconstruimos la distribución:

```
cp ipmitool-1.8.11-1.ia32e.rpm \
/state/partition1/home/install/rocks-dist/lan/x86_64/RedHat/RPMS
cd /state/partition1/home/install
rocks-dist dist
```

Y simplemente reinstalamos los nodos con:

```
ssh-agent $SHELL
ssh-add
tentakel -g compute '/boot/kickstart/cluster-kickstart-pxe'
```

## 3.3. Instalación vía NFS

Nuevamente partimos del código fuente de la librería a configurar e instalar. Sin embargo, no construiremos el RPM, pues hay algunas librerías con las que este procedimiento tiene una mayor complejidad. Y para sortear la creación del paquete RPM procederemos a distribuir los ficheros necesarios por NFS.

Como indicamos anteriormente, la librería en este caso será MUMPS. Será distribuida por NFS, usando un directorio compartido como punto de instalación (`/share/apps/`); este directorio será exportado automáticamente a los nodos.

Por tanto, procedemos a descargar el software en el directorio `/share/apps/` y lo descomprimimos.

Para configurar esta librería necesitamos generar un archivo de configuración llamado `Makefile.inc`. En un directorio de la instalación (`Make.inc`) disponemos de múltiples ficheros de configuración genéricos que nos servirán a modo de ejemplo. En este preciso caso trabajaremos sobre el Makefile relacionado con los compiladores de Intel como veremos posteriormente.

```
cp Make.inc/Makefile.Intel.PAR Makefile.inc
nano Makefile.inc
```

En dicho archivo de configuración especificaremos los comandos correspondientes a los compiladores de Intel, en su versión paralela (`mpiicc` y `mpiifort`), y las librerías de Intel oportunas. Preferimos incluir estas librerías debido a que poseen un mayor rendimiento comparándolas con otras librerías. De esta forma es más probable que podamos detectar ineficiencias en el software que desarrollaremos. El fichero `Makefile.inc` queda como el que mostramos en fichero 3.4.

Nótese que el entorno debe estar correctamente configurado, en concreto, las variables con los *paths*. Ello se hace mediante *modules*; en este caso, cargado el módulo correspondiente a los compiladores de Intel junto con Intel MPI (*impi* en nuestro caso). Una vez configurada correctamente la construcción de la librería, debemos construirla e instalarla en el *frontend*. De este modo, debemos teclear en el terminal:

```
module add impi
make
```

Los nodos dispondrán de la librería a través del servicio NFS, y en este caso las variables de entorno para nodos y *frontend* coinciden (en esta librería).

Es importante, tener en cuenta que estamos ante varios ordenadores, con distintas funciones: uno compilará, otro ejecutará... Y las variables de entorno deben

**Fichero 3.4: Makefile.inc para MUMPS**

```
LPORDDIR = $(topdir)/PORD/lib/
IPORD    = -I$(topdir)/PORD/include/
LPORD    = -L$(LPORDDIR) -lpord

LMETISDIR = $(HOME_METIS)
IMETIS    = # Metis doesn't need include files (Fortran interface avail.)
LMETIS    = -L$(LMETISDIR) -lmetis

ORDERINGSF  = -Dpord -Dmetis
ORDERINGSC  = $(ORDERINGSF)
LORDERINGS  = $(LMETIS) $(LPORD)
IORDERINGS  = $(IMETIS) $(IPORD)

PLAT    =
RM      = /bin/rm -f
CC      = mpiicc
FC      = mpiifort
FL      = mpiifort
AR      = ar vr
RANLIB  = echo

INTEL_DIR  = /opt/intel
SCALAP_DIR = $(INTEL_DIR)/mkl/10.0.1.014
MPI_DIR    = $(INTEL_DIR)/mpi/3.1
SCALAP  = -lmkl_scalapack_lp64          \
          -lmkl_blacs_intelmpi20_lp64   \
          -lmkl_intel_lp64              \
          -lmkl_intel_thread            \
          -lmkl_core                    \
          -lguide

INCPAR    = -I$(MPI_DIR)/include64
LIBPAR    = -L$(SCALAP_DIR) $(SCALAP)
LIBOTHERS = -lpthread
CDEFS     = -DAdd_
OPTF    = -O2 -nofor_main
OPTL    = -O2 -nofor_main
OPTC    = -O2
INC = $(INCPAR)
LIB = $(LIBPAR)
```

ser consistentes de un ordenador a otro, para que los programas ejecuten correctamente. Rocks está pensado para acceder solamente al *frontend*, la fase de compilación se realiza en el *frontend* con unas variables de entorno concretas. Si cambiamos las localizaciones de las variables de entorno en los nodos, deberemos ser consecuentes cuando ejecutemos el programa. La modificación de las variables de entorno, sólo en los nodos, resulta ligeramente complicada, y no es práctica. Nosotros tratamos de evitar este punto recreando la estructura de directorios del *frontend* en los nodos, y usando *modules*.

## 3.4. Instalación local en los nodos.

En los nodos podemos instalar la librería en cualquier directorio, pero, como hemos dicho, deberemos modificar el entorno de trabajo para adaptarlo a la situación, por lo que recomendamos que la instalación en los nodos sea en el mismo directorio que en el *frontend*, como en nuestro caso.

Instalaremos MUMPS en el *frontend* en `/opt/`. Eso lo hacemos de la misma forma que hemos visto en el apartado anterior 3.3 pero cambiando el directorio de instalación (ahora es `/opt/` en vez de `/share/apps/`). Tras este paso, y procederemos a completar la instalación en los nodos. La instalación en los nodos consiste en copiar las librerías del *frontend*, en el directorio elegido para la instalación. Para ello hacemos uso del servicio NFS, concretamente de un directorio con permisos (`/share/apps/`).

Creamos el directorio que distribuirá la instalación y comprimimos los archivos necesarios:

```
mkdir -p /share/apps/MUMPS
cd /opt
tar -czf /share/apps/MUMPS/libs.tar MUMPS_4.8.4/*
```

Generalmente, sólo nos interesa distribuir las librerías, pero en este caso, por simplicidad, la distribuiremos completa. En el archivo de configuración de los nodos (`/home/install/ site-profiles/4.3/nodes/extend-compute.xml`), que hemos visto en la instalación de RPMS, contiene una zona de postinstalación. Esta zona ejecuta comandos una vez realizada la instalación, que utilizaremos para descomprimir y copiar los archivos necesarios. Editamos el fichero y escribimos en la zona habilitada para tal efecto:

**Fichero 3.5: extend-compute.xml de la distribución**

```
<post>
  /bin/mount -t nfs -r \
    :/state/partition1 /mnt
  /bin/mkdir -p /opt/MUMPS_4.8.4
  cd /opt/MUMPS_4.8.4 ;
  /bin/tar -xzpf /mnt/apps/MUMPS/libs.tar.gz
  /bin/umount /mnt
</post>
```

Aunque usemos un directorio que se exporta por NFS a los nodos, debemos recordar que el nodo no está instalado, es decir, carece de usuarios y servicios, por lo que deberemos montar el directorio (en este caso en `/mnt`). Ahora debemos reconstruir la distribución; siempre que toquemos los ficheros de configuración debemos construirla para que los cambios tengan efecto, y reinstalar los nodos.

```
cd /state/partition1/home/install
rocks-dist dist
cluster-fork '/boot/kickstart/cluster-kickstart'
```

Con lo que hemos terminado, y podemos comprobar en cualquier nodo si se ha instalado o no simplemente haciendo:

```
ssh compute-0-0 ls /opt/MUMPS_4.8.4
```

Si todo ha funcionado correctamente, debemos ver lo mismo que haciendo `ls /opt/MUMPS_4.8.4` desde el *frontend*.

## 3.5. Construir una Roll

### 3.5.1. Fase de análisis

El primer paso es analizar el problema, teniendo siempre presentes los objetivos. A continuación, listamos el software que deseamos instalar:

- Intel® Compiler Suite Professional Edition for Linux, incluye tanto los compiladores de C++ y Fortran, como Intel® Threading Building Blocks y Intel® Integrated Performance Primitives, en su versión 11 y sólo para x86_64.
- En esta versión versión de los compiladores se incluye la Intel® Math Kernel Library (Intel® MKL) for Linux. Sin embargo, incluiremos la versión 10 completa (para todas las arquitecturas), dado que posee funciones especificas que podrían resultarnos de interés.

Ahora debemos considerar que sólo el *frontend* deberá disponer de una instalación completa (librerías, ejecutables, ...), y los nodos solamente deberán tener las librerías dinámicas, aquellas necesarias en tiempo de ejecución. Así pues, la instalación en los nodos será diferente a la instalación del *frontend*. Además necesitaremos la versión em64t, pues todos nuestros nodos se pueden englobar en esta categoría.

Ahora bien, para generar una Roll, tenemos que tener muy presente que está basada en paquetes RPMs, y por lo tanto, el caso que presenta mayor complejidad, es el caso de construir una Roll a partir de código fuente. Para evitar tener que generar a mano el RPM, Rocks dispone de un sistema automático, que consiste

en un conjunto de Makefiles y estructuras, como hemos visto anteriormente en la instalación de RPMs. Esta herramienta genera los ficheros necesarios para la creación del paquete RPM, y lo construye. Después utilizará este paquete RPM para la creación de la Roll.

Para realizar la Roll, necesitamos realizar una instalación de prueba para comprobar cual debe ser su localización y si necesitaremos otros ficheros o no. Como el desarrollo de Rolls y la instalación de las Rolls pueden generar problemas de estabilidad en el *frontend*, optamos por recrear dos *frontenis* y uno o dos nodos en máquinas virtuales. Uno de los *frontenis* lo destinamos a desarrollo, mientras que el otro se destina a pruebas de instalación.

En el *frontend* de desarrollo, instalamos los compiladores y la MKL, y analizamos la instalación, donde descubrimos que el proceso de instalación completa se puede reducir a instalar unos RPMs ya construidos por Intel:

- intel-cproc110083e-11.0-1.x86_64.rpm
- intel-cproc110083iidbe-11.0-1.x86_64.rpm
- intel-cproc110083ipp32e-11.0-1.x86_64.rpm
- intel-cproc110083mkl32e-11.0-1.x86_64.rpm
- intel-cproc110083tbb32e-11.0-1.x86_64.rpm
- intel-cprof110083e-11.0-1.x86_64.rpm
- intel-cprof110083iidbe-11.0-1.x86_64.rpm
- intel-cprof110083mkl32e-11.0-1.x86_64.rpm
- intel-mkl101024-10.1p-024.noarch.rpm

Una vez instalados los paquete RPM listados anteriormente, necesitaremos incluir en el entorno las variables correspondientes. Dado que utilizamos *modules* generaremos el módulo correspondiente (Fichero 3.6), analizando los ficheros `iccvars.sh` e `ifortvars.sh` que contienen las variables de entorno que necesitamos (en esta versión del módulo no se introducen las variables correspondientes a la MKL 10). Este fichero lo añadiremos con la Roll-Restore, al igual que el archivo de licencia necesario para usar los compiladores.

El último punto que debemos conocer y entender es el grafo de la distribución. El grafo dirige el proceso de instalación y configuración de manera dinámica. Como veremos más adelante el grafo es capaz de distinguir los nodos y aplicar los

**Fichero 3.6: Módulo intel11**

```
#%Module1.0#####################################################################
    ###
##
## modules modulefile
##
## modulefiles/intel11.  Generated from modules.in by root.
##
proc ModulesHelp { } {

        puts stderr "\tIntel Compiles & MKL module"
        puts stderr "\n\tThis adds Intel11"
        puts stderr "\tenvironment variables."
}

module-whatis   "loads the Intel 11 environment"

# for Tcl script use only
set     version         11
set     dir             /opt/intel/Compiler/11.0/083/
#idb
set     NLSPATH $dir/idb/intel64/locale/%l_%t/%N
#tbb
set TBB21_INSTALL_DIR $dir/tbb
set TBB_ARCH_PLATFORM /em64t/cc3.4.3_libc2.3.4_kernel2.6.9
prepend-path LIBRARY_PATH $dir/tbb/em64t/cc3.4.3_libc2.3.4_kernel2.6.9/lib
prepend-path LD_LIBRARY_PATH $dir/tbb/em64t/cc3.4.3_libc2.3.4_kernel2.6.9/lib
prepend-path DYLD_LIBRARY_PATH $dir/tbb/em64t/cc3.4.3_libc2.3.4_kernel2.6.9/
    lib
prepend-path CPATH $dir/tbb/include
#mkl
set MKLROOT /opt/intel/Compiler/11.0/083/mkl
prepend-path INCLUDE $MKLROOT/include
prepend-path  LD_LIBRARY_PATH $MKLROOT/lib/em64t
prepend-path  MANPATH $MKLROOT/man/en_US
prepend-path  LIBRARY_PATH $MKLROOT/lib/em64t
prepend-path  CPATH $MKLROOT/include
prepend-path  FPATH $MKLROOT/include
prepend-path  NLSPATH $MKLROOT/lib/em64t/locale/%l_%t/%N

prepend-path PATH $dir/bin/intel64
prepend-path  LD_LIBRARY_PATH $dir/lib/intel64
prepend-path  NLSPATH $dir/lib/intel64/locale/%l_%t/%N
set INTEL_LICENSE_FILE /opt/intel/licenses
prepend-path MANPATH $dir/man
```

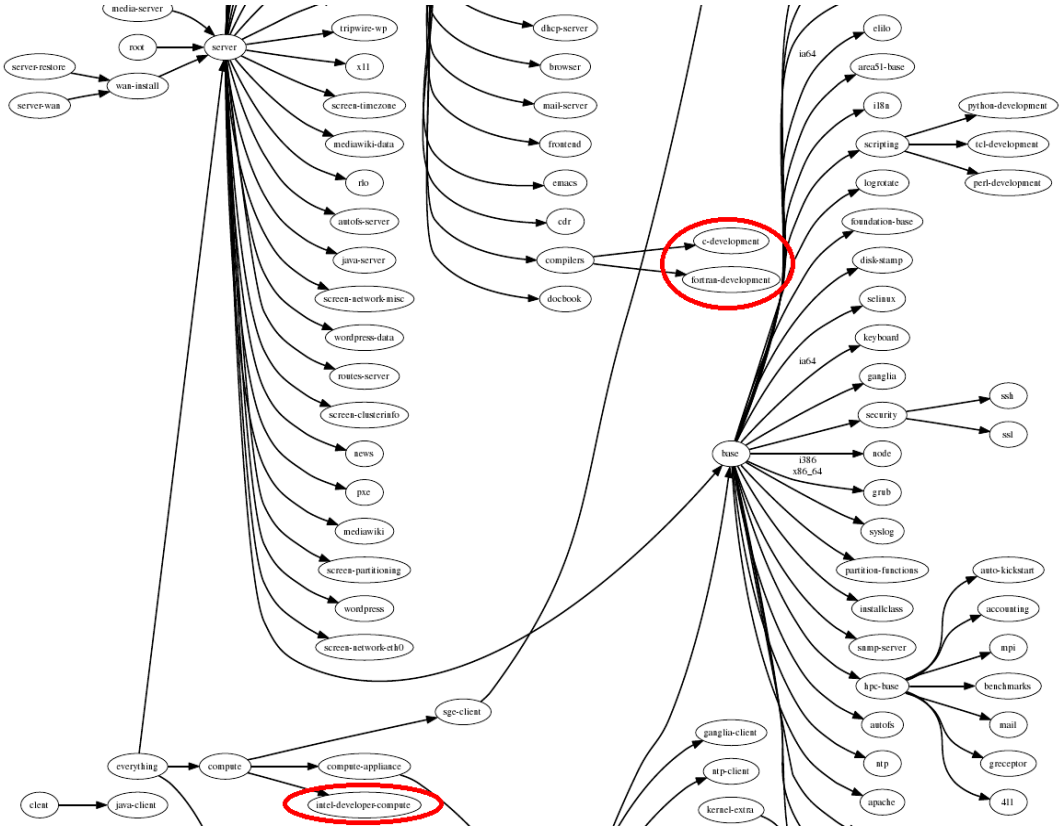



Figura 3.1: Grafo de nuestra distribución.

paquetes pertinentes. Nosotros añadiremos conexiones y nodos al grafo, rara vez introduciremos cambios estructurales como veremos al terminar este proceso. Podemos ver en la figura 3.1 un fragmento del grafo simplificado y en rojo podemos ver la Roll *Intel Development*, tanto para el *frontend* como para los nodos.

El grafo está descrito en XML. En él se indica el nodo y las conexiones en un fichero, las tareas a realizar en otro fichero. Como veremos, es un procedimiento sumamente flexible. En este caso los nodos c-development y fortran-development, corresponden a la Roll Intel Development desarrollada por ClusterCorp, que contienen los compiladores de Intel en su versión 10. Cada Roll posee su propio grafo, y la distribución genera el grafo agregado, así que podemos describir un grafo más pequeño, sin necesidad de conocer completamente el grafo de la distribución. Sólo deberemos conocer los nodos básicos de la distribución:

- **server** se refiere al *frontend*.
- **client** a los nodos.

- **base** tanto al *frontend*, como a los nodos.

### 3.5.2. Construcción de la Roll

La realizaremos en dos pasos, primero generaremos la instalación en el *frontend* y posteriormente en los nodos. Comenzaremos moviéndonos a la carpeta de trabajo en el *frontend*:

```
cd /state/partition1/site-roll/rocks/src/roll/
```

Y creamos el árbol de directorios, similar al que generaríamos para hacer un paquete RPM:

```
/state/partition1/site-roll/rocks/src/roll/bin/\
make-roll-dir.py -n intel -v 1 -c blue
```

En este caso bautizamos la nueva Roll como intel (con el parámetro `-n`, que indica el nombre), además será la versión 1 (con `-v`) y en el grafo estará representado con color azul (`-c blue`).

#### Configuración de la instalación en el *frontend*.

Ahora creamos el directorio `RPMS` con dos subdirectorios `noarch` y `x86_64`. Y copiamos los paquetes RPM anteriormente descritos en estos directorios, los terminados en `noarch.rpm` en la carpeta `noarch` y los `x86_64.rpm` en `x86_64`. Sólo necesitamos crear el archivo XML con el grafo para finalizar configuración de la instalación en el *frontend*.

El grafo es sencillo, de dos nodos básicos (server y client) parten dos conexiones a otros dos nodos, uno para el *frontend* y otro para los nodos. Queda reflejado en `graphs/default/intel.xml` de la siguiente manera:

**Fichero 3.7: graphs/default/intel.xml**

```
<edge from="server">
        <to>intel</to>
</edge>
<edge from="client">
        <to>intel-extend</to>
</edge>
```

Cuando ponemos `<to>intel</to>`, hacemos referencia a otro fichero XML situados en la carpeta `nodes`, llamado `intel.xml`, que especifica que acciones tendrán lugar en el nodo del grafo (*edge*) en cuestión. En este caso del nodo del grafo *server* (que hace referencia al *frontend*) cuelga el nodo *intel*, y las acciones que tendrán lugar se especifican en `nodes/intel.xml`. Y de la misma manera de *client* cuelga *intel-extend*, y las acciones están descritas en `nodes/intel-extend.xml`

En este fichero, `graphs/default/intel.xml`, donde describimos el grafo podemos especificar un orden de instalación, como vemos a continuación:

**Fichero 3.8: Ejemplo con orden de instalación**

```
<order head="intel">
  <tail>intel-extend</tail>
</order>
```

Con lo que la configuración de la instalación del *frontend* se realizaría antes que la de los nodos.

Y para concluir la configuración de la instalación en el *frontend* debemos escribir el fichero `nodes/intel.xml` que deberá contener todos los RPMs de la siguiente manera:

**Fichero 3.9: nodes/intel.xml**

```
  <package arch="x86_64">intel-cproc110083e</package>
2 <package arch="x86_64">intel-cproc110083e</package>
  <package arch="x86_64">intel-cproc110083iidbe</package>
  <package arch="x86_64">intel-cproc110083ipp32e</package>
  <package arch="x86_64">intel-cproc110083mkl32e</package>
  <package arch="x86_64">intel-cproc110083tbb32e</package>
7 <package arch="x86_64">intel-cprof110083e</package>
  <package arch="x86_64">intel-cprof110083iidbe</package>
  <package arch="x86_64">intel-cprof110083mkl32e</package>
  <package>intel-mkl101024</package>
```

Con esto queda configurada la instalación de los paquetes que necesitábamos.

Es en este punto donde podemos especificar la arquitectura, tal y como vemos en el ejemplo, e incluir archivos. Por ejemplo, el fichero 3.6, descrito anteriormente, podríamos incluirlo así:

**Fichero 3.10: Ejemplo con escritura de fichero**

```
  <file name="/root/intel11" mode="append" perms="666">
  #%Module1
      0#######################################################################
  ##
  ## modules modulefile
5 ##
```

```
## modulefiles/intel11.  Generated from modules.in by root.
 [...]
prepend-path MANPATH $dir/man
</file>
```

En la primera línea le damos nombre al fichero, modo de escritura y permisos y a continuación escribimos el fichero.

Y realizar operaciones post-instalación añadiendo `<post>commands</post>`.

### Configuración de la instalación en los nodos.

Para realizar la configuración de la instalación en los nodos, deberemos crear un RPM, y nos ayudaremos de las herramientas de Rocks (como anteriormente para instalar un RPM), para ello nos dirigimos al directorio `src/intel` y creamos dos carpetas: una con las librerías de los compiladores `lib`, y otra con las correspondientes a la MKL `mkl`. Y generamos un *intel-1.tgz*, aunque la extensión se puede cambiar, como vimos en la instalación de RPMS.

```
cd src/intel
mkdir lib mkl
cp -r /opt/intel/mkl/10.1.2.024/lib mkl/
cp -r /opt/intel/Compilers/11.0/083/lib lib/
mkdir lib/idb lib/mkl lib/ipp lib/tbb
cp -r /opt/intel/Compilers/11.0/083/idb lib/idb
cp -r /opt/intel/Compilers/11.0/083/mkl lib/mkl
cp -r /opt/intel/Compilers/11.0/083/ipp lib/ipp
cp -r /opt/intel/Compilers/11.0/083/tbb lib/tbb
tar -czf intel-1.tgz mkl lib
```

Nótese que para poder hacer lo anterior se debe contar con una instalación ya en el frontend. Debemos tener el software instalado y configurado para nuestra arquitectura, y después procederemos con la configuración de la instalación en los nodos.

Y a continuación modificamos el fichero `Makefile` de ese mismo directorio para que realice las operaciones oportunas, es muy importante tener claro que la generación del RPM se realiza en dos pasos: instalación (*build*) y construcción(*install*):

**Fichero 3.11: /src/intel/Makefile**

```
PKGROOT = /opt/intel
REDHAT.ROOT     = $(PWD)/../../
ROCKSROOT       = ../../../../..
-include $(ROCKSROOT)/etc/Rules.mk
include Rules.mk
build:
        tar -zxf $(NAME)-$(VERSION).$(TARBALL_POSTFIX)
install::
```

```
        mkdir -p $(ROOT)/$(PKGROOT)/Compilers/11.0/083
        mkdir -p  $(ROOT)/$(PKGROOT)/mkl/10.1.2.024/
        cp -rp lib/*  $(ROOT)/$(PKGROOT)/Compilers/11.0/083
        cp -rp mkl/* $(ROOT)/$(PKGROOT)/mkl/10.1.2.024/
clean::
        rm -rf $(NAME)-$(VERSION)
```

Y para terminar volvemos al grafo, completando el fichero `nodes/intel-extend.xml`. Nótese que, dado que el RPM es generado automáticamente, su nombre será "intel".

```
<package arch="x86_64">intel</package>
```

Para terminar la configuración de la instalación, retocamos el fichero del directorio raíz llamado `version.mk`, añadiendo que el tamaño de la imagen creada sea cero:

**Fichero 3.12: version.mk del directorio principal**

```
ISOSIZE = 0
RELEASE = 0
COLOR   = blue
REDHAT.ROOT = \$(PWD)
```

Hemos forzado el tamaño de la imagen a cero, porque excede los 700 MB, que es el tamaño por defecto de la imagen de CD-ROM, y de esta manera Rocks calculará el tamaño de manera automática. Si omitimos este punto, la imagen creada tendrá el valor por defecto y no será viable.

Y construimos la Roll con:

```
make roll
```

Al final debemos obtener algo así:

```
intel-1-0: 9b228216f15576667bddfd1269787640
roll-intel-kickstart-4.3-0: 29c0f1a2686c1cb91ec0a0fa039b8b67
Creating disk1 (0.00MB)...
Building ISO image for disk1 ...
[root@ash25 intel]# ls -la *.iso
-rw-r--r--  1 root root 1392855040 May 18 05:18 intel-4.3-0.x86_64.disk1.iso
```

### Ayudas a tener en cuenta

- Rehacer una Roll sin reconstruir los RPMs:

  Si algún RPM falla de la Roll en desarrollo, en lugar de volver a construirla podemos hacer:

  ```
  make reroll
  ```

Para reconstruir, hay que tener cuidado de no hacer `make clean` o borrar el directorio de RPMs o SRPMs.

- Verificación del grafo y nodos XML:

  Un pequeño script para verificar la sintaxis de xml antes de construir la Roll es el que sigue:

**Fichero 3.13: Script de verificación**

```
\#!/opt/rocks/bin/python

import os
from xml.dom.minidom import parse
\# graph
for file in os.listdir('graphs/default') :
        if not file.endswith('.xml') :
                continue
        print 'parsing ' + file
        path = os.path.join('graphs/default', file)
        parse(path)
\# node
for file in os.listdir('nodes') :
        if not file.endswith('.xml') :
                continue
        print 'parsing ' + file
        path = os.path.join('nodes', file)
        parse(path)
```

Podemos ejecutarlo añadiendo esto al Makefile del directorio raíz:

**Fichero 3.14: Modificaciones al Makefile.**

```
verify:
        ./xml\_verify
default: verify roll
```

- Comprobación del RPM que contiene el grafo de la Roll:

  La Roll tiene un RPM llamado *roll-kickstart-[roll name]* que contiene el grafo y los nodos de la Roll. Para agilizar la comprobación, podemos comprobar la salida de estos comandos (en el directorio de desarrollo de la Roll):

```
#crea roll-kickstart-<roll name>.rpm en RPMS/<arch>
make profile
rpm -Uvh RPMS/<arch>/roll-kickstart-<roll name>.rpm
cd /home/install && rocks-dist dist
# now test the graph & nodes file
rocks list host profile compute-0-0
rocks list host profile <\textit{frontend} name>
```

- Comprobar errores en la generación:

  Uno de los problemas de construir una roll es que no haya errores críticos mientras se genera. Como normalmente a pesar de los errores, el makefile continua ejecutándose, es conveniente comprobar esos errores de la siguiente forma:

```
make roll >& log < /dev/null &
```

Los posibles errores los podemos encontrar comprobando la salida del siguiente comando:

```
cat roll | grep -i error | egrep -v 'checking|ignored|\
gcc|ld|ar' | less
```

### 3.5.3. Comprobación de la instalación

La comprobación de la instalación puede hacerse de dos formas:

- Instalación sobre la distribución, partiendo del `.iso` generado.

  Montamos la ISO y la incluimos en la distribución:

  ```
  mount -o loop intel-4.3-0.x86_64.disk1.iso /mnt/cdrom
  rocks-dist copyroll
  ```

  Desmontamos el CD-ROM y construimos la distribución:

  ```
  umount /mnt/cdrom
  cd /home/install
  rocks-dist dist
  ```

  Finalmente la activamos con:

  ```
  rocks enable roll intel
  ```

- Instalación completa en el *frontend* de pruebas: Siguiendo los pasos detallados en el anexo B, incluimos la nueva Roll en tiempo de instalación inicial del `frontend`. Debemos comprobar que obtenemos una pantalla similar a la mostrada en la figura 3.2, donde comprobamos que se está instalando uno de los RPMs que hemos incluido en la Roll. O posteriormente podemos entrar en el *frontend* y en el nodo y comprobar que efectivamente se encuentran allí.

También podemos ver el grafo de la distribución y encontraremos los nuevos nodos como podemos ver en las figuras 3.3 y 3.4.

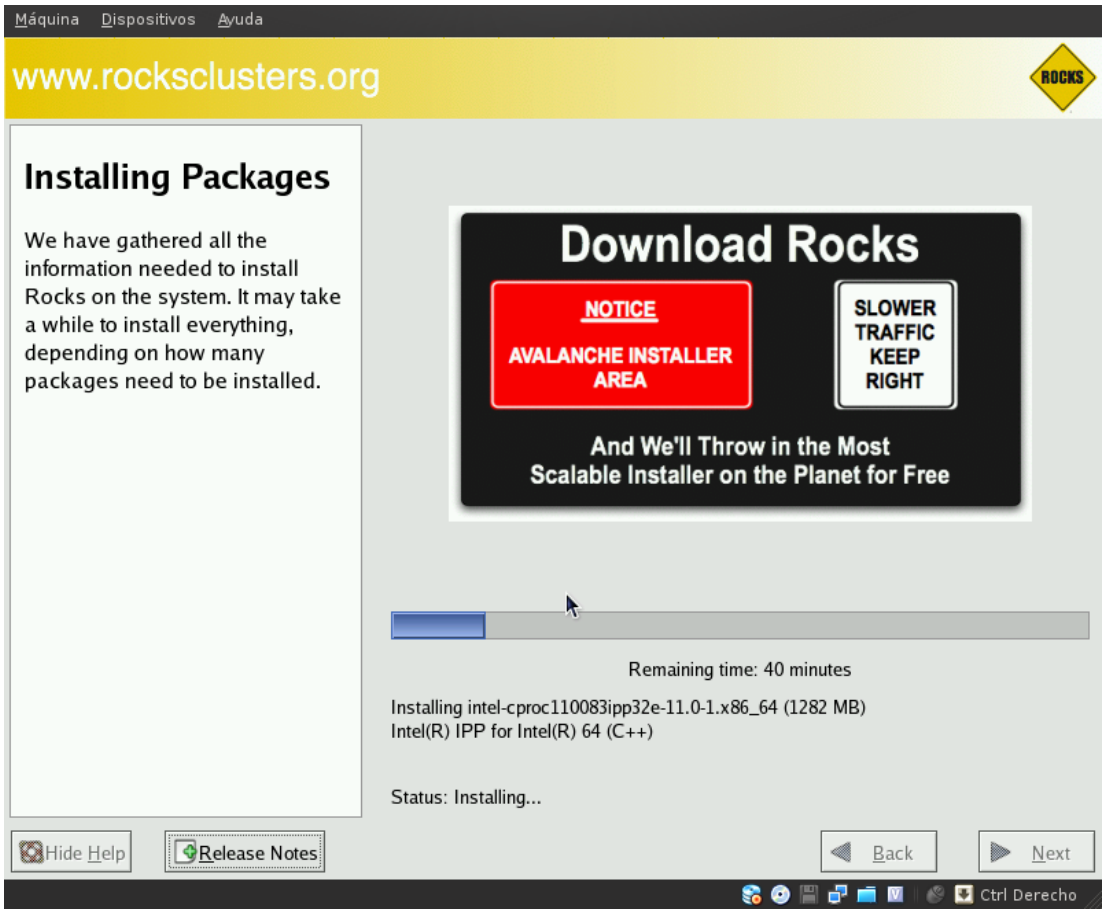



Figura 3.2: Instalación de la Roll.

## 3.6. Conclusiones

Existen multitud de métodos para integrar software en Rocks, pero si queremos que los cambios perduren en el tiempo deberemos confeccionar una Roll. Crear Rolls puede ser un proceso complejo, y no requiere una fase de pruebas.

Cada uno de los métodos estudiados tiene su propia razón de ser, dependiendo del software a instalar y el tiempo que dispongamos para ello. Por ejemplo si no disponemos de tiempo y necesitamos una librería realizaremos una instalación por NFS, pero si disponemos de suficiente tiempo, desarrollaríamos una Roll.

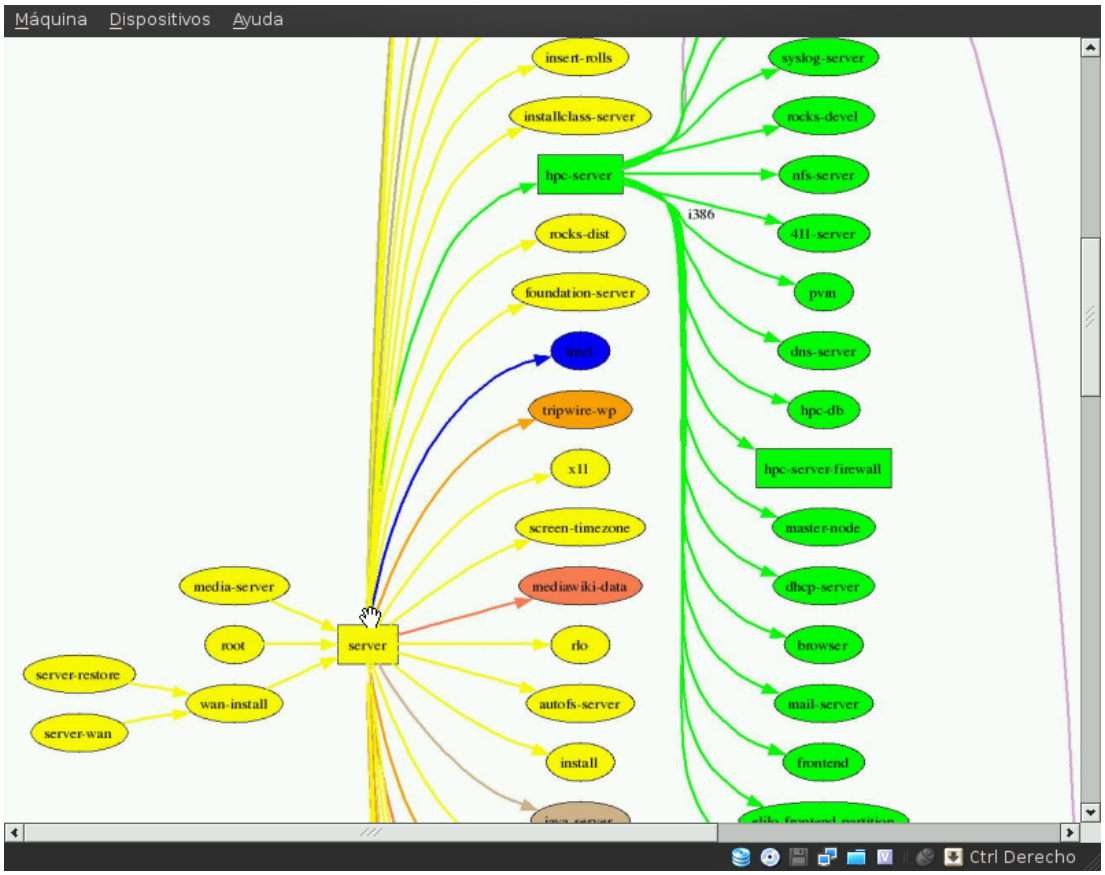



Figura 3.3: Nuevo grafo de nuestra distribución I.

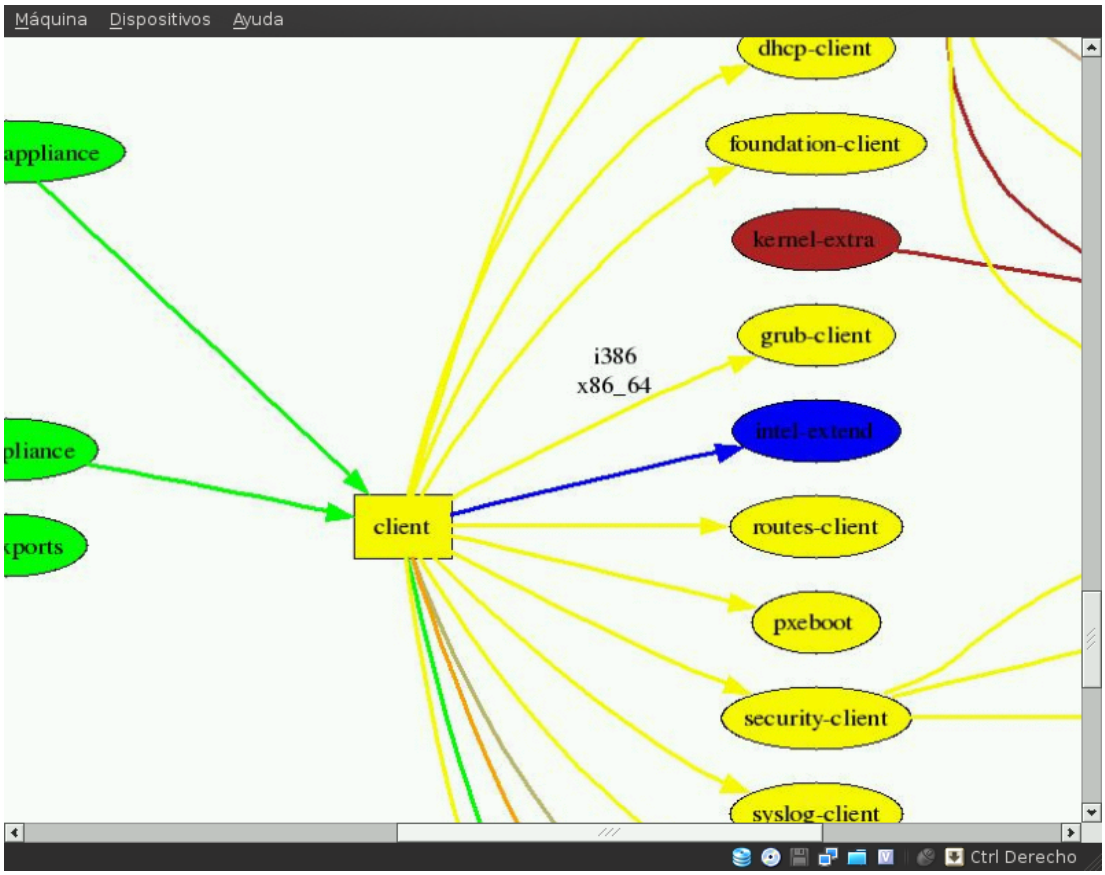



Figura 3.4: Nuevo grafo de nuestra distribución II.

# Capítulo 4

# Pruebas de rendimiento.

En el presente capítulo presentamos las pruebas realizadas con el fin de estudiar el comportamiento del cluster ante diferentes problemas y estrategias de resolución. El comportamiento del cluster, dependerá de cómo son tratados los diferentes paradigmas que presenta la arquitectura del cluster.

En este preciso caso tenemos varios paradigmas, teniendo en cuenta la figura 4.1:

- El primero un CC-UMA, *Cache Coherent Uniform Memory Access*, en cada procesador, dado que internamente implementan más de un core y una sola cache para todos ellos. Cada acceso a memoria de cada core es legible por todos los cores del chip. Este paradigma se clasifica como de memoria compartida.
- El segundo que se nos presenta es la presencia, en cada placa, de dos procesadores, en este caso sería NUMA (*Non Uniform Memory Access*), es decir, podemos acceder con un procesador a los datos del otro debido a que comparten memoria RAM. Este paradigma, también, se clasifica como de memoria compartida.
- El tercero, tal vez el más evidente, es de memoria distribuida ya que todos los nodos de nuestro cluster están conectados por red.

La ventaja de la memoria compartida es la rapidez y facilidad para pasar información entre cores/procesadores, la gran desventaja es la escalabilidad, pues quedan limitados por el ancho de banda de la memoria (los accesos a memoria de este tipo de máquinas son el principal factor limitante).

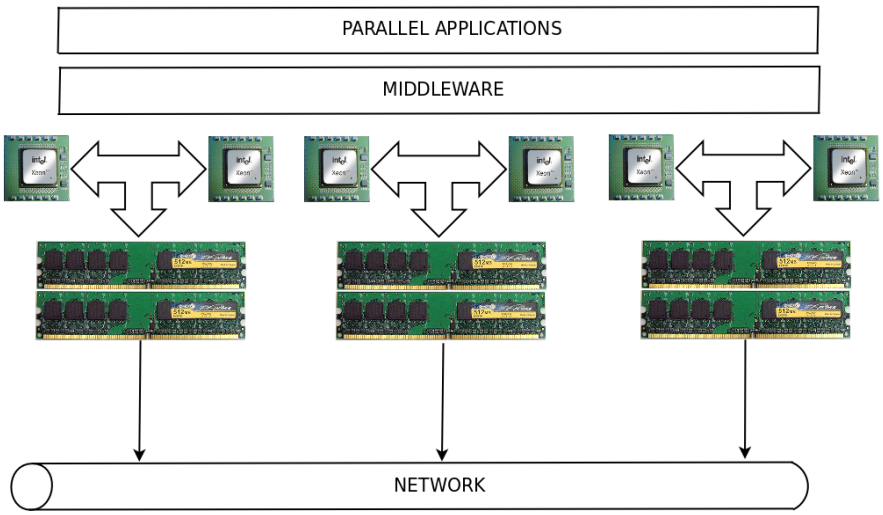



Figura 4.1: Arquitectura de un cluster.

Las ventajas de la memoria distribuida son la escalabilidad, que es lineal en memoria y procesadores, y que cada procesador puede acceder a su memoria sin ninguna limitación. Pero las limitaciones son la estructuración de la memoria, que queda en manos del programador el intercambio de información, y nuevamente el ancho de banda de la red. Por poner un ejemplo, en nuestro caso el ancho de banda entre memoria y procesador es de 10,6 Gbps e incluso podríamos duplicar esta cifra, pero el interfaz de red tiene 1 Gbps (un orden de magnitud de diferencia, sin hablar de los retardos en los tiempos de accesos).

En nuestro caso el primer paradigma viene dado por el hardware, trataremos de aprovecharlo, pero no como objetivo principal. Evitaremos el segundo paradigma con las estructuras de datos, separando las regiones de memoria de cada proceso. Trabajaremos sobre el tercero con la ayuda de *PETSc* (*Portable, Extensive Tookit for Scientific Computation*). La aproximación para tratar el paradigma de memoria compartida es el paso de mensajes y usaremos MPI como vimos en el capitulo anterior.

Las pruebas a las que someteremos a nuestro cluster serán cuatro:

- Un test de rendimiento genérico, donde emplearemos *Linpack* [28]. Con este test comprobaremos el funcionamiento general del cluster.
- Haremos pruebas de escalabilidad con un resolvedor paralelo iterativo que desarrollaremos con *PETSc*. Podemos encontrar más información sobre *PETSc* en [29] y en [30].

- Probaremos un resolvedor multifrontal paralelo basado en MUMPS [27] en el contexto de matrices de elementos finitos para electromagnetismo implementado en el código denominado "HibridoFEM3D". El código HibridoFEM3D implementa diversos métodos híbridos (en el sentido de combinación de FEM y métodos asintóticos de alta frecuencia) de análisis de problemas electromagnéticos. En esta Proyecto Fin de Carera hacemos sólo uso de la parte FEM y, en concreto, estamos interesados en el rendimiento del resolvedor multifrontal que incorpora.
- Por último comprobaremos el funcionamiento de un simulador de problemas de dispersión y radiación basado en el Método de los Momentos, cuyo resolvedor paralelo trabaja con matrices densas, [31].

Debemos tener en cuenta que la prueba de rendimiento genérico es prácticamente teórica, el resto de pruebas corresponden a algoritmos reales resolviendo problemas electromagnéticos reales.

## 4.1. Test de Linpack: Rendimiento general.

Debemos comprobar que el cluster tiene una potencia de cálculo similar a la teórica. Esta prueba detecta problemas de rendimiento en las comunicaciones y el hardware.

El cálculo teórico de la capacidad de computo del cluster se realiza sumando para cada nodo la multiplicación de el número de operaciones por ciclo de reloj, por la frecuencia del reloj, por el número de cores que posee el procesador, y por el número de procesadores por nodo. En nuestro caso, el número de operaciones por ciclo de reloj es cuatro, y el número de procesadores es dos, para todos los nodos. Sólo operaremos con los esclavos, por lo que hablamos de 481,3 Gflops como máximo teórico.

Para medir la capacidad de cómputo del cluster empleamos el test de Linpack, desarrollado en el Argone National Laboratory, que es uno de los usados en el campo de la supercomputación (forma parte del conjunto de test utilizado para calcular el TOP 500 [32]).

El principio básico del test consiste en medir el tiempo empleado en la ejecución de ciertas instrucciones. En su mayoria son llamadas a BLAS (Basic Linear Algebra Subprograms, [33]), que realiza operaciones matriciales y vectoriales optimizando el número de datos de entrada, y está organizada en tres niveles:

- El primer nivel de BLAS trata todas las operaciones que involucran escalares y vectores.
- El segundo nivel se encarga de las operaciones matriz por vector.
- En el tercer nivel de BLAS se realizan las operaciones matriciales, como los productos matriz por matriz.

Son varias las instrucciones que se miden en el test, concretamente mide las instrucciones necesarias para:

- resolver un sistema 100x100, $[A]\mathbf{x} = \mathbf{b}$ con $[A]$ aleatorio,
- un sistema 1000x1000 como el anterior,
- un problema paralelo escalable,

este test nos dará una medida muy importante sobre el estado y funcionamiento de nuestro cluster.

Para ejecutarlo deberemos generar un fichero de entrada, como el fichero 4.1.

Las dos primeras líneas carecen de importancia, ya que son completamente ignoradas por el programa de test. Las dos siguientes lineas determinan el nombre del fichero y donde se volcará la información, en este caso hemos elegido que vuelque la información a un fichero llamado HPL.out.

La quinta linea describe cuantos problemas resolveremos. Este número debe ser menor o igual a 20 y mayor que cero, el tamaño de los problemas se describe en la linea siguiente, donde deberá haber $N$ números enteros. El tamaño debe ser inferior a la raíz cuadrada de la suma de la memoria de los nodos que participan en el test. En nuestro caso, para evitar problemas tomamos la memoria de todos los nodos, como el número de cores multiplicado por la mínima relación de memoria por core, de esta forma evitamos saturar cualquiera de los nodos que participan en el test.

Las líneas 7 y 8 determinan la granularidad de los mensajes intercambiados. Deben estar en el intervalo [32,256]. Nuestras pruebas determinaron que obteníamos el máximo para una granularidad de 160, por lo que fijamos este valor en el test.

Las tres siguientes líneas determinan el grid de procesos, la primera el número de grids que probaremos y las dos siguientes los tamaños del grid. Como disponíamos

**Fichero 4.1: HPL.in configuración de Linpack.**

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
8            device out (6=stdout,7=stderr,file)
2            # of problems sizes (N)
15000 18000       Ns
1            # of NBs
160           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
2            # of process grids (P x Q)
23 2          Ps
2  23         Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) f
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
##### This line (no. 32) is ignored (it serves as a separator). ######
0                               Number of additional problem sizes for PTRANS
30000           values of N
0                               number of additional blocking sizes for
    PTRANS
40 20 16 32 64 128 160          values of NB
```

de 46 cores teníamos cuatro posibles grids, 1-46, 46-1, 2-23 y 23-2, descartamos 1-46 y 46-1, debido a que se recomiendan grids cuadrados.

La línea decimotercera determina el umbral de comparación para los residuos. En este caso dejamos el valor por defecto, ya que, intervienen diversos cálculos, y pese a poder marcar los test como fallidos, obtendremos igualmente el resultado de capacidad.

Desde la decimocuarta hasta la vigésimo primera linea, se describen características especificas del algoritmo, que dejamos por defecto. Las dos lineas siguientes describen los comunicadores, en los que usamos un método llamado *ring Modified*.

Las lineas vigésimo cuarta y vigésimo quinta describen cuantos conjuntos son factorizados después de actualizarlos. En este caso se recomienda que sea 1, por

lo que nuevamente dejamos el valor por defecto. La lineas vigésimo sexta versa sobre el algoritmo de intercambio utilizado en todo el test, en este caso elegimos *spread-roll*. La siguiente linea carece de importancia debido a que no se usa en el test. Las dos lineas que siguen tratan sobre el almacenamiento de las matrices, elegimos la forma transpuesta en ambos casos, debido a que es más eficiente.

La trigésima linea sólo se tiene en cuenta si en la vigésimo sexta se elige 1 o 2, con lo que en este caso está plenamente activa, y elegimos añadir una etapa que balancee la carga, debido a que tenemos cores con diferente capacidad de cómputo. La siguiente linea especifica el alineamiento en memoria para los doubles, en este caso 8 para estar seguros. Los resultados obtenidos fueron muy satisfactorios, el cluster alcanza los 480 Gflops reales:

**Fichero 4.2: Resultados obtenidos con Linpack.**

```
Column=000160 Fraction=0.005 Mflops=480236.86
```

Los resultados son operaciones en coma flotante de doble precisión por segundo, esto es que es capaz de realizar 480 miles de millones de sumas y multiplicaciones cada segundo.

## 4.2. Resolvedor iterativo con *PETSc*.

*PETSc* ([30]), es una biblioteca diseñada para proyectos de cálculo científico, programada bajo C, C++, Fortran y Python; que incluye una gran cantidad de rutinas y estructuras de datos, para trabajar tanto secuencialmente, como de forma paralela, con PDEs (*Partial Differential Equations*).

*PETSc* nos permite programar sobre MPI de forma simple y eficaz, proporciona:

- Vectores paralelos, tanto densos como dispersos.
- Matrices paralelos con varios formatos para matrices dispersas.
- Precondicionadores paralelos escalables.
- Métodos de subespacios de Krylov.
- Resolvedores paralelos no lineales, basados en métodos de Newton.
- Resolvedores paralelos ODE (*Ordinary Differential Equations*).

- Una documentación muy completa.
- Tratamiento automático de la memoria.
- Interfaz consistente, con comprobación sistemática de los errores.
- Posee muchos ejemplos.
- Con soporte y proyección para muchos años.

Además de poder tratar MPI a alto nivel, podremos hacer uso de librerías matemáticas de alto rendimiento, como BLAS y LAPACK. LAPACK (Linear Algebra PACKage) está escrita en Fortran 77, y proporciona algoritmos para realizar operaciones con matrices, como la factorización LU o Cholesky, pero no trata matrices dispersas, aunque si está optimizada para entornos SIMD (*Single Instruction Multiple Data*).

En nuestro caso, usaremos la librería MKL de Intel, Math Kernel Library, que nos proporcionará las dos anteriores, además de ScaLAPACK (Cubre la deficiencia de LAPACK en cuanto a matrices dispersas), optimizadas para los procesadores de Intel.

Como podemos ver en la arquitectura de la librería (figura 4.2), nuestro código se situará a alto nivel, y para incluir la librería MKL de Intel lo único que deberemos hacer es referenciarla en la etapa de configuración. Una vez compilemos *PETSc* podremos hacer uso de todas las estructuras y funciones que incorpora.

En nuestro caso la configuración se realizó de la siguiente manera:

**Fichero 4.3: Configuración de PETSc.**

```
cd petsc-2.3.2-p10
config/configure.py --with-cc=icc --with-fc=ifort --with-cxx=icpc --with-mpi-
    include=/opt/intel/mpi/3.1/include --with-mpi-lib=/opt/intel/mpi/3.1/lib/
    libmpich.a --with-blas-lapack-dir=/opt/intel/mkl/10.0.1.014/lib/em64t/ --
    with-shared=0 --with-clanguage=C++ --useThreads=0 --configModules=PETSc.
    Configure --with-fortran-kernels=1 --with-scalar-type=complex --with-
    debugging=0
make all test
```

Con esta configuración, contamos con los compiladores, y librerías MPI y MKL de Intel, activando algunas opciones para mejorar el rendimiento, como evitar que el programa hiciera algún tipo de debug o que por defecto use las funciones en Fortran (recordemos que las principales librerías de alto rendimiento están escritas en este lenguaje).

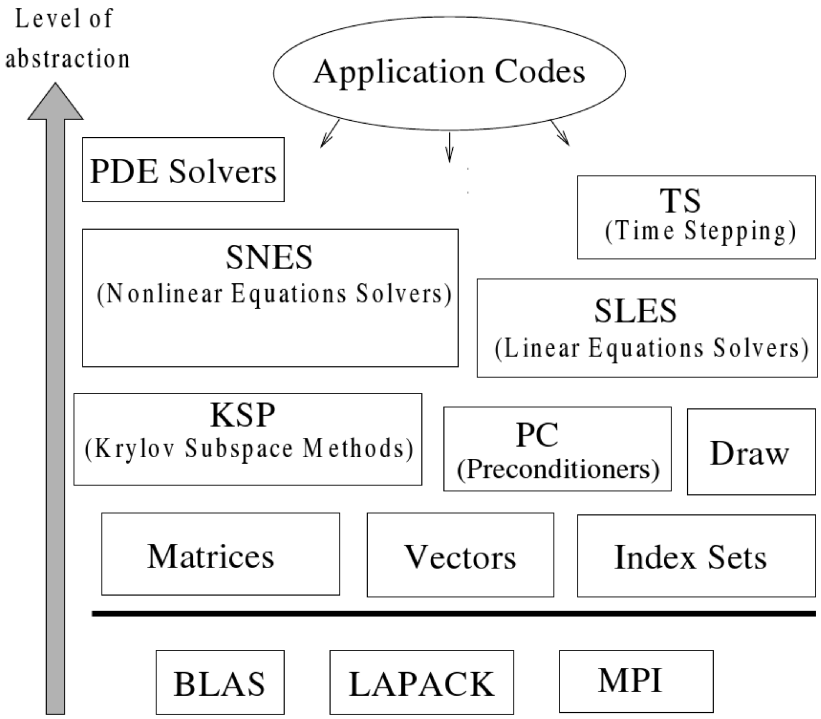



Figura 4.2: Arquitectura de *PETSc* .

El modelo de programación con *PETSc* se orienta a la portabilidad y el paralelismo escalable, y permite cambiar de arquitectura sin tener que recodificar el programa; para ello deberemos compilar la librería para esa arquitectura.

En una primera aproximación trataremos de incluir un resolvedor paralelo en un programa ya existente. El programa, llamado HibridoFEM3D nos proporcionará todos los elementos necesarios para generar el sistema de ecuaciones. Pero nos enfrentaremos a dos inconvenientes:

- Diferencias de lenguajes: pese a que con *PETSc* podemos escribir en Fortran, esta escrito en C y HibridoFEM3D está escrito en Fortran. Esto nos generará problemas con el tamaño de las variables en memoria.

- Orientación: *PETSc* no esta orientado a ser una función o una subrutina de otro programa, es decir, está diseñado para ser el núcleo donde incluimos otras librerías y subrutinas.

Ambos inconvenientes están muy ligados, ya que, si empleásemos *PETSc* como programa principal y agregásemos las subrutinas no tendríamos ninguno de los dos inconvenientes, porque *PETSc* se encargaría de dimensionar las variables. Pero usar *PETSc* como núcleo requiere recodificar todo el programa, en este caso HibridoFEM3D, y como sólo buscamos agregar un resolvedor descartamos esa opción. En 4.2.1 veremos como solventar estos inconvenientes.

### 4.2.1. Interfaz C/Fortran

Lo primero que necesitamos saber son los pasos que damos al compilar el código fuente, las fases y subfases de cualquier compilador son:

- Análisis
  - Léxico
  - Sintáctico
  - Semántico
- Síntesis
  - Código intermedio
  - Código objeto
  - Optimización (este paso no es obligatorio, pero es deseable)

Además debemos saber que existen otras herramientas asociadas al compilador, como son:

- Preprocesador, consiste en un etapa de análisis léxico y síntesis, anterior a la acción del compilador, con el fin de facilitar la labor de este. Trata de expandir macros, e incluir ficheros.
- Linker, generalmente los programas no se generan de una sola vez, se forman de pequeñas porciones de código objeto, y es el linker el que las une y genera el fichero ejecutable.

Generalmente al llamar al compilador automáticamente llama al preprocesador, realiza la compilación, y después invoca al linker.

Las opciones que tenemos para solventar estos problemas, son:

**Fichero 4.4: ejef-c.F**

```
! File: ejef-c.F
      program main
      implicit none
      external hola
      call hola
      end
      subroutine hi
      implicit none
%       write(*,*) "Hola mundo!"
      end subroutine
```

**Fichero 4.5: ejec-f.c**

```
# File: ejec-f.c
#include <stdio.h>
extern int hi_();
void hola_(){
printf("Hello world!\n");
}
int main(){
hi_();
return 1;
}
```

- La segunda opción es utilizar `Cfortran.h` (Un fichero de definiciones) diseñado por los autores de Fortran, que nos proporciona un increíble número de macros y opciones que simplifican el trabajo, a la hora de compilar. Si bien tenemos que estudiar la documentación y puede llegar a sobrecargar el preprocesador.

- Otra opción es utilizar todos los nombres en minúsculas, debido a la sintaxis de Fortran y C, y hacer uso de alias como, por ejemplo,

  ```
  !DEC$ ATTRIBUTES ALIAS:'My_Proc_' :: My_Proc
  ```

  Con lo que en la etapa de preprocesado conseguiremos que ambos lenguajes se comuniquen eficientemente.

En nuestro caso seguimos esta última opción analizando la naturaleza de los problemas que se nos presentan para proporcionar interoperabilidad entre estos lenguajes.

Veamos las llamadas a Fortran desde C y al revés, con dos ficheros 4.4 y 4.5.

La única diferencia aparente es que la cadena de caracteres que imprimen por pantalla esta en español y en inglés respectivamente. Las llamadas a Fortran desde

C, darán como resultado `Hola mundo` y podemos comprobarlo de la siguiente manera:

```
ifc -c ejef-c.F          #Genera el archivo ejef-c.o
icc -c ejec-f.c          #Genera el archivo ejec-f.o
ifc -o eje1 ejec-f.o ejef-c.o #Genera el ejecutable eje1
```

Al ejecutar el programa podemos comprobar que es `Hola mundo` y tenemos que tener en cuenta que `program main` ha de ser comentado del archivo `ejef-c.F` debido a las dobles referencias a main.

Las llamadas a C desde Fortran se realizan de la misma forma comentando el main del archivo `ejec-f.c` la salida que hemos obtenido es:

```
./eje1
Hello world!
```

Es decir que para llamar desde Fortran a C, que es lo que vamos a necesitar, basta con añadir un guión bajo al final de la definición de la función para que Fortran pueda llamarlo. Si C llama a Fortran, tenemos que colocar este guión bajo. Existen opciones de compilación para añadir estos guiones de manera automática.

**Otros puntos a tener en cuenta**

- Paso de parámetros a las funciones: Fortran utiliza referencias (punteros) para pasar las variables por lo que desde C al llamar a Fortran se han de pasar punteros y al diseñar las funciones en C para ser llamadas por Fortran deben utilizar punteros.

- Arrays: Fortran utiliza un almacenamiento basado en columna mientras que C lo hace en filas, la matriz desde Fortran a C ha de ser traspuesta y viceversa. Si esto no se hace, el programa funciona, pero lo hace mal (para evitar transponer la matriz lo que se puede hacer es intercambiar los índices teniendo en cuenta que en Fortran se comienza en 1 y en C en 0). Lo recomendable es definir unas macros como las siguientes:

  ```
  #define B(i,j) b[j-1][i-1]
  #define B(i) b[i-1]
  ```

  con lo que la trasposición y los índices quedan automáticamente corregidos en el preprocesado.

- Variables: hay que tener en cuenta que el tamaño de las variables varía con el lenguaje:

| FORTRAN | C |
|---|---|
| BYTE | unsigned char |
| INTEGER*2 | short int |
| INTEGER*4 | int |
| REAL*4 | float |
| REAL*8 | double |
| COMPLEX*8 | struct complex float r, i; |
| INTEGER | int |
| CHARACTER*n | char x[n] * |
| INTEGER x(m) | int x[m] |
| REAL x(m) | float x[m] |
| CHARACTER*n x(m) | char x[m][n] * |

- Strings: En Fortran las longitudes de los strings se pasan por valor (no es un puntero) como argumento oculto justo después del string. Esto no aparece en los argumentos de las funciones de Fortran, no obstante ha de aparecer en las de C.

### 4.2.2. Programando con *PETSc*

Para la realización de nuestro resolvedor paralelo tenemos como requisito imprescindible la coexistencia con otros resolvedores de ecuaciones. Y para gestionar este punto, sabiendo que además no es conveniente una elección dinámica del resolvedor, elegimos desarrollar una pasarela (en la literatura anglosajona podemos encontrarlo como *wrapper*) mediante directivas de precompilación, comandadas por el Makefile. De esta forma evitamos la dependencia de librerías asociadas a los resolvedores de los que no hagamos uso, no requiere disponer de todas las librerías usadas en los diferentes resolvedores, sólo de aquellas que el resolvedor en cuestión necesite.

Veamos un pequeño fragmento de la pasarela:

**Fichero 4.6: Pasarela para varios resolvedores**

```
MODULE Solver
#if SOLVER_HSL
        USE SolveHSL
#endif
#if SOLVER_MUMPS
        USE SolveMUMPS
#endif
#if SOLVER_PETSC
        USE SolvePETSC
#endif
subroutine FinalizaSolver
#if SOLVER_MUMPS
```

```
13          if(id%myid.eq.0) then
                    deallocate(id%eltptr)
                    deallocate(id%eltvar)
                    deallocate(id%a_elt)
                    deallocate(id%rhs)
18          end if
            id%job=-2
            call zmumps(id)
    #endif
    #if SOLVER_PETSC
23          call FinalizaPETSC
    #endif
    end subroutine FinalizaSolver
    !..........!
```

Está aproximación minimiza las decisiones dinámicas, y maximiza la memoria en la primera etapa de análisis y síntesis en el precompilador, donde se desechará todo el código y librerías de los resolvedores que no usemos, pasando a la siguiente etapa solamente el código que necesitamos.

Esto además nos da una gran ventaja, pues aún desarrollando uno de los resolvedores, el código es perfectamente usable. Esto es muy deseable, el programa gana modularidad y pierde dependencias, es decir, no hace falta disponer, ni instalar las librerías que no usemos.

La desventaja es que el código resulta menos legible, ya que deberemos diferenciar las directivas de preprocesamiento de los comentarios, por lo que tratamos de resolverlo usando la pasarela como hemos visto, encapsulando las diferentes rutinas para disponer de un interfaz limpio a un nivel superior. Lo más deseable en este punto, es que la pasarela tenga el menor número de lineas de código posible, por lo que una vez realizada y depurada, no deberemos modificarla, salvo para añadir otro resolvedor.

En la figura 4.3 podemos ver una estructura muy simplificada de HibridoFEM3D, donde incluiremos nuestro resolvedor paralelo.

Básicamente el algoritmo que rige HibridoFEM3D, consiste en resolver sucesivamente un sistema de ecuaciones cambiando la excitación (el segundo miembro).

Nos centraremos en el desarrollo del resolvedor (la caja *Sparse Solver* en la figura 4.3). En un principio nuestro resolvedor iterativo para matrices dispersas, resolverá un sistema genérico donde el segundo miembro cambia. Veremos la estructura del resolvedor en detalle en la figura 4.4.

Todas las subrutinas se encapsularon en un módulo, escrito en Fortran con formato libre, pues, como ya hemos dicho, pese a que *PETSc* mitiga la tarea de

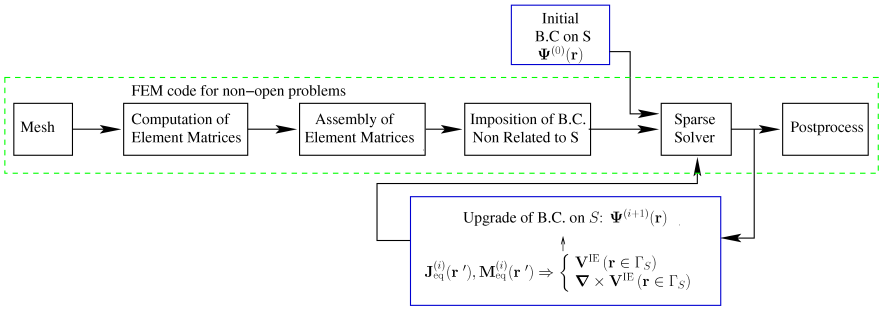



Figura 4.3: Diagrama de flujo simplificado de HibridoFEM3D.

escribir las pertinentes adaptaciones entre Fortran y C, nos será mucho más fácil en este formato.

A continuación iremos desgranando poco a poco el programa y los elementos clave que lo componen, comenzando por los *includes* y macros:

**Fichero 4.7: Includes y macros definidas en el resolvedor.**

```
  !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  !                      Include files
  !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#include "include/finclude/petsc.h"
#include "include/finclude/petscvec.h"
#include "include/finclude/petscmat.h"
#include "include/finclude/petscksp.h"
#include "include/finclude/petscpc.h"
 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  !                    Macro declarations
  !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  !este macro define la interfaz entre fortran y c para los indices de los
      vectores
#define get(ib)  ((ib)-1)
#define ADD_VALUES 2
#define INSERT_VALUES 1
```

Por orden el primero de los *includes* nos dará acceso a las variables y definiciones propias de *PETSc* para Fortran (incluidos alias para las funciones, como vimos anteriormente). En nuestro caso usaremos vectores y matrices deslocalizados, es decir, ningún nodo tendrá la información completa de un vector o una matriz, salvo que lo programemos expresamente. Nuestro resolvedor paralelo será iterativo, y haremos uso de subespacios de Krylov[1] y algún precondicionador[34].

[1]Krylov SubsPace, dada una matriz,$[A]$, de $NxN$ y un vector,$\mathbf{b}$, $N$-dimensional, es el espacio vectorial generado por imagenes de $\mathbf{b}$ en primeras $r$ potencias de $[A]$, esto es:

$$K_r([A], \mathbf{b}) = span\{[A]^k \mathbf{b} \quad \forall k \in \{0, \ldots, r-1\}\}$$

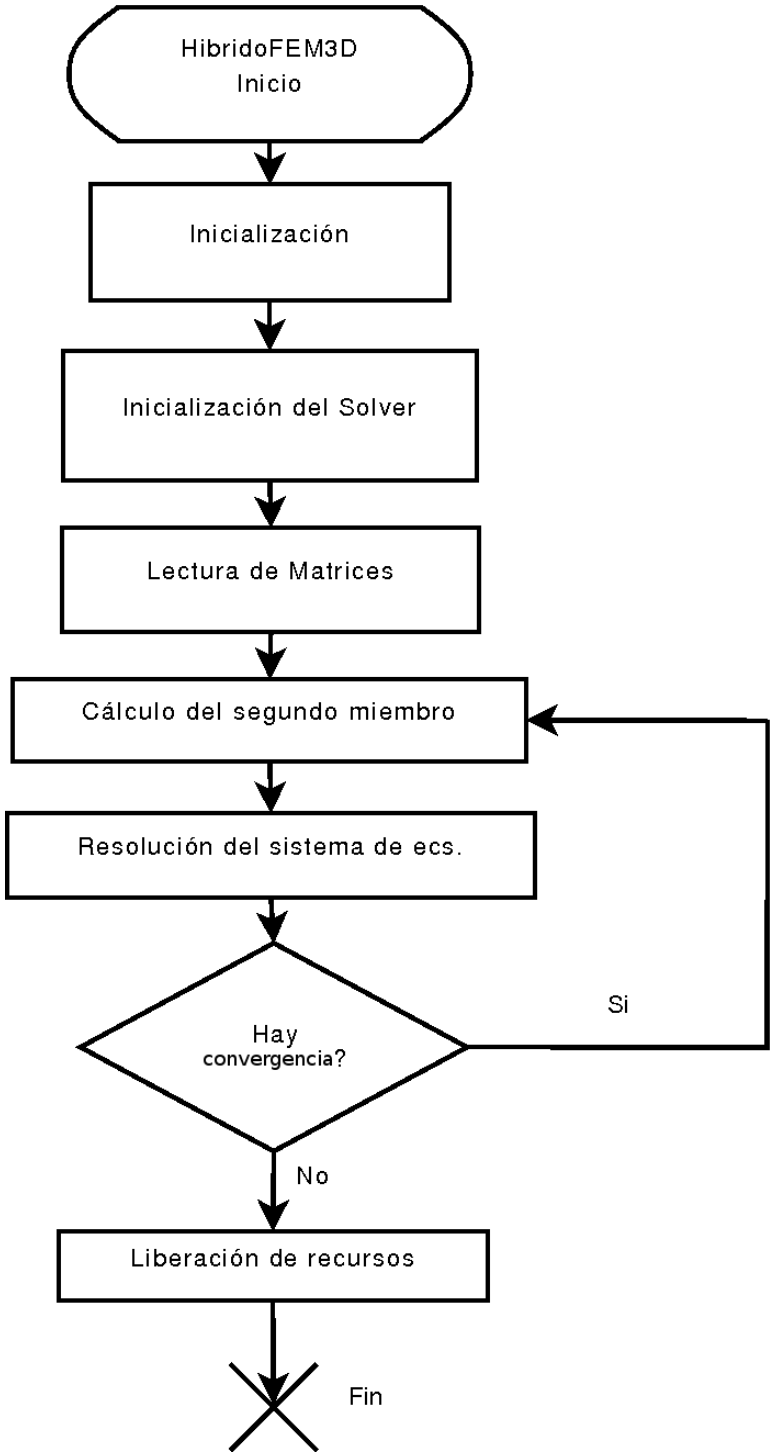



Figura 4.4: Diagrama de flujo del resolvedor de HibridoFEM3D.

Las macros están definidas para facilitar el paso de los datos entre C y Fortran, y para asegurar el valor de dos variables, pues en la versión en la que hemos desarrollado, se definían con tipo de datos, y al programar para em64t surgió el problema del tipo de datos, del que hemos hablado anteriormente.

**Fichero 4.8: Declaración de variables del resolvedor.**

```
!  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                    Variable declarations
!  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        Mat             A_
        Vec             B_global, X_global, X_local
        PetscOffset     idx
        PetscInt        xsize, low, high,  reason, its
        PetscScalar     xx(1)
        Integer         Msize, Mrank, pierr, aa, bb, low_e, high_e
        VecScatter      scatterGLOBAL
        KSP             ksp
        PC              pc
```

En las variables anteriores, podemos ver la matriz y vectores del sistema $[A]\mathbf{x} = \mathbf{b}$, pero lo que es más importante, vemos un `X_global,X_local` esto es debido a que nuestro módulo resolvedor necesitará tener el vector solución íntegro en un nodo, para poder devolver el valor al programa principal. *PETSc* está pensado para ser la base de desarrollo, no para ser una librería auxiliar. Por este motivo, su uso en este punto es complicado.

**Fichero 4.9: Inicialización de las estructuras.**

```
subroutine InicializaPETSC
!  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                    Beginning
!  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        call PetscInitialize(PETSC_NULL_CHARACTER, Pierr)
        call MPI_Comm_rank(PETSC_COMM_WORLD, Mrank, Pierr)
        call MPI_Comm_size(PETSC_COMM_WORLD, Msize, Pierr)
        call MatCreateMPIAIJ(PETSC_COMM_WORLD, PETSC_DECIDE, PETSC_DECIDE,&
          Datos%Nrnodos, Datos%Nrnodos, 200, PETSC_NULL_INTEGER, 190, &
          PETSC_NULL_INTEGER, A_, PETSCierr)
        call MatSetFromOptions(A_, Pierr)
        call MatZeroEntries(A_, pierr)
        call VecCreate(PETSC_COMM_WORLD, x_Global, Pierr)
        call VecSetSizes(x_global, PETSC_DECIDE, Datos%Nrnodos, Pierr)
        call VecSetFromOptions(x_global, Pierr)
        call VecDuplicate(x_global, b_global, pierr)
        call VecGetOwnershipRange(X_global, low, high, pierr)
        call KSPCreate(PETSC_COMM_WORLD, ksp, pierr)
end subroutine InicializaPETSC
```

Vemos la inicialización de *PETSc* , que sólo puede ser realizada una vez, es decir dos llamadas a `PetscInitialize`, producirán fugas de memoria. Dentro de este inicializador, se inicializa MPI, con las siguientes llamadas tenemos el número del proceso y el número total de ellos.

La matriz utilizada será dispersa con el formato AIJ[2], del resto de llamadas, tal vez debemos resaltar las terminadas en `SetFromOptions` que nos permitirán alterar todos los parámetros desde la linea de comando una vez compilado el código.

**Fichero 4.10: Liberación de memoria.**

```
1 subroutine FinalizaPETSC
          !....!
          call VecDestroy(B_GLOBAL, pierr)
          call VecDestroy(x_global, pierr)
          call VecDestroy(x_local, pierr)
6         call MatDestroy(A_, pierr)
          call PetscFinalize(pierr)
  end subroutine FinalizaPETSC
```

El módulo resolvedor está constituido de tres partes:

- Inicialización del subespacio de Krylov, con el precondicionador.
- El ensamblado de la matriz, pues la matriz $[A]$ aún no existe de forma completa.
- La propia resolución, con la devolución del resultado.

En un principio se precondicionó el sistema por la propia matriz $[A]$, lo que propicia que la matriz sea semi-definida positiva, y, en consecuencia, mejora la convergencia. Pero debido a la formulación empleada, nuestro módulo resolvedor demostró ser divergente. Este problema se puede solucionar de tres formas: cambiando la formulación por una formulación estabilizada, diseñando un precondicionador ex profeso, o buscando un precondicionador algebraico de propósito general que haga converger el método iterativo. Exploramos la tercera opción, buscando entre los precondicionadores que incorpora *PETSc* y encontramos que un precondicionador de SSOR, hacia que la sucesión de los $\mathbf{x}^k$ fuera convergente, pero no era el precondicionador óptimo, ya que el coste computacional era descomunal. Una vez asegurada la convergencia, empleamos varios métodos iterativos, como BiGC o GMRES, y el método que obtuvo la convergencia más rápida fue GMRES. Por lo que las opciones por defecto de nuestro resolvedor paralelo serán:

- Precondicionador de SSOR.
- Resolvedor de Mínimo RESiduo Generalizado (GMRES).

[2]El formato AIJ también es llamado *Yale sparse matrix format* o *compressed row storage*.

Y nuevamente recordar que estos parámetros se pueden variar al ejecutar el programa, en la línea de comando.

El ensamblado de la matriz se realiza de forma paralela, es decir se divide el número de elementos de la matriz entre los procesos, y se calcula por separado la matriz correspondiente a cada elemento. Esto corresponde a la formulación integral local. El cálculo de la forma integral global consiste en sumar todas las matrices elementales, es decir, se realiza en `MatSetValues` y, para terminar, deberemos asegurarnos que el estado de la matriz es operativo, y eso lo hacemos con la pareja de llamadas `MatAssemblyBegin` y `MatAssemblyEnd`.

**Fichero 4.11: Ensamblado de las matrices.**

```
        call own(datos%Nrelem,mrank,msize,low_e,high_e)
2       do elemento=low_e,high_e!1,datos%Nrelem
          !llamadas a librer as para el calculo de la matriz del elemento
          call MatSetValues(A_,20,nodos(:,elemento)-1,20,nodos(:,elemento)-1,
              element_matrix,ADD_VALUES,Pierr)
        enddo
        call MatAssemblyBegin(A_,MAT_FINAL_ASSEMBLY,Pierr)
7       call MatAssemblyEnd(A_,MAT_FINAL_ASSEMBLY,Pierr)
```

Ahora que disponemos de la matriz, tenemos configurado e inicializado el resolvedor, sólo queda resolver el sistema de ecuaciones y pasar el resultado al programa principal en un vector de Fortran. Como se puede ver en el siguiente fragmento de código, introducimos el segundo miembro del sistema, resolvemos, y después creamos un comunicador global, para pasar el vector global (deslocalizado entre los procesos) a uno "local" que contenga todo el contenido del vector. Y después simplemente recogemos el resultado.

**Fichero 4.12: Obtención del resultado.**

```
  subroutine SolveWithPETSC
        PetscScalar              puntero(0:Datos%Nrnodos-1)
3       Integer                  pto(0:Datos%Nrnodos-1)
          !...!
        call VecSetValues(B_global,high-low,pto,puntero, INSERT_VALUES,pierr)
        call vecassemblybegin(B_global,pierr)
        call vecassemblyend(B_global,pierr)
8       call KSPSolve(ksp,B_global,x_global,pierr)
        call VecScatterCreateToAll(X_global,scatterGlobal,x_local,pierr)!
            creates scatter &vector
        call VecScatterBegin(X_global,x_local,INSERT_VALUES,SCATTER_FORWARD,
            scatterGlobal,pierr)
        call VecScatterEnd(X_global,x_local,INSERT_VALUES,SCATTER_FORWARD,
            scatterGlobal,pierr)
          !...!
13      call  VecGetValues(x_local,Datos%Nrnodos,pto,puntero,PETSCierr)
        call VecScatterDestroy(scatterGlobal,pierr)
  end subroutine SolveWithPETSC
```

Como dijimos anteriormente, el módulo resolvedor era plenamente funcional, y conseguía convergencia, pero debido a la formulación empleada, la matriz está mal

condicionada y la convergencia era excesivamente lenta. Para evitar que la matriz estuviera mal condicionada pasamos a comprobar el funcionamiento de la librería con matrices conocidas, garantizando una buena convergencia.

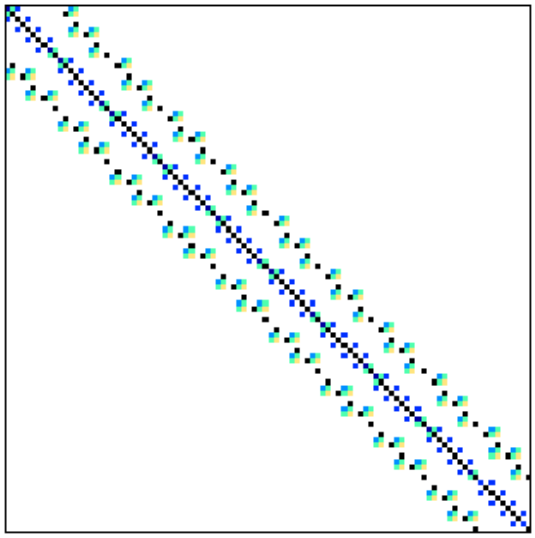
Las matrices de prueba empleadas, fueron escogidas de Matrix-Market[3], y se codificó un nuevo programa, que incluye una rutina para la lectura de matrices. El programa es idéntico, prescindiendo del ensamblado, y añadiendo la rutina secuencial para la lectura de la matriz. La rutina de lectura fue reescrita partiendo de la librería que proporciona Matrix-Market, integrando el ensamblado de la matriz, es decir, cada proceso leerá la matriz completa e incorporará los elementos que necesite. Paralelizar la lectura a disco, pese a ser muy deseable, nos fue imposible implementarla, y al integrar la lectura en serie con el ensamblado de la matriz en memoria, no podremos acotar que tiempo corresponde a la lectura en serie. Esto mejora la velocidad en líneas generales pero cuando tengamos muchos procesos, los resultados no serán tan buenos (reducimos la pendiente de la curva de *speedup*). Escogimos las matrices pensado en comprobar además como escala el problema con el tamaño de la matriz.

La estructura de las matrices que se pueden ver en las figuras 4.5,4.6 y 4.7, fueron seleccionadas por ser semi-definidas positivas, reales y simétricas, siendo obtenidas en su mayoría del método de elementos finitos. Sus principales características son:

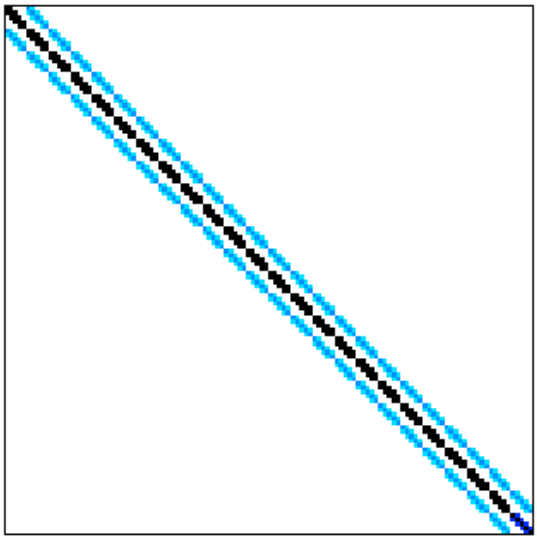
- Nos 4 es una matriz dispersa, de 100x100, con 594 elementos distintos de cero.
- Nos 3, es una matriz dispersa, de 960x960, con 15844 elementos distintos de cero.
- Bloweybq, es una matriz dispersa, de 10001x10001, con 49999 elementos distintos de cero.
- 2cubes_sphere, es una matriz dispersa, de 101492x101492, con 1647264 elementos distintos de cero.
- Ecology2, es una matriz dispersa, de 999999x999999, con 4995991 elementos distintos de cero.

Para resolver estas matrices se creo un vector aleatorio, para que al multiplicarlo nos diera el segundo miembro. Expresado de otra manera, sea **S** un vector

[3]es un repositorio de matrices posee cerca de 500 matrices dispersas, así como generadores de matrices[35]

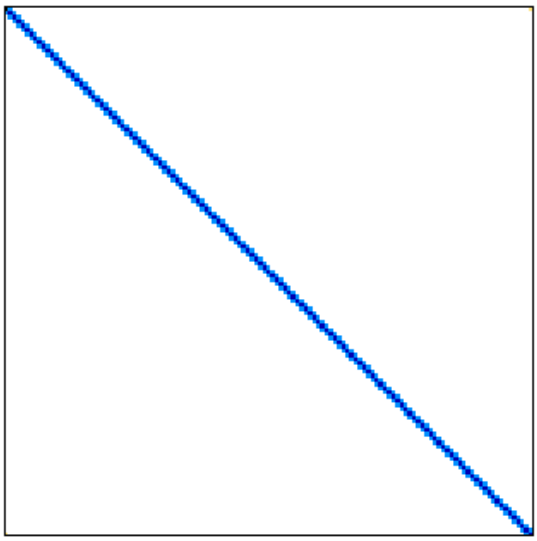


(a) Matriz Nos4

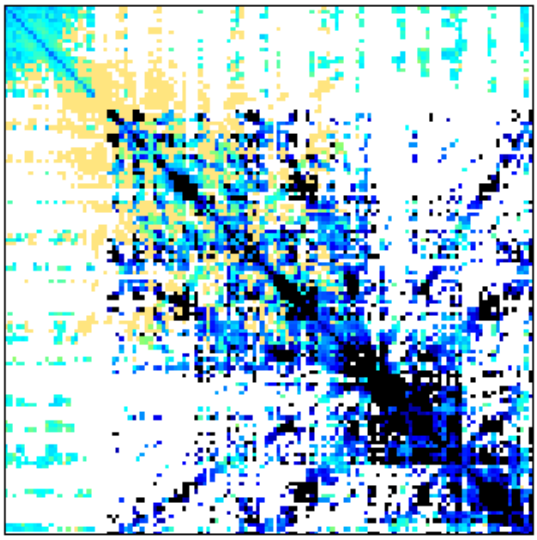


(b) Matriz Nos3

Figura 4.5: Aspecto de las matrices seleccionadas I.

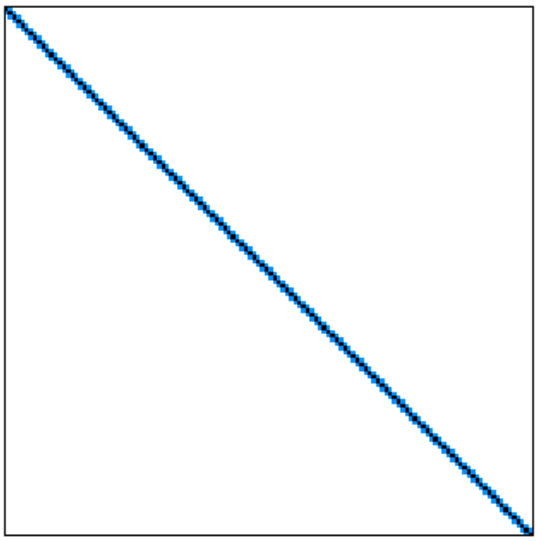


(a) Matriz Bloweybq



(b) Matriz 2cubes_sphere

Figura 4.6: Aspecto de las matrices seleccionadas II.



(a) Matriz Ecology2

Figura 4.7: Aspecto de las matrices seleccionadas III.

aleatorio (conocido), entonces $\mathbf{b} = [A]\mathbf{S}$, y procedemos a resolver $[A]\mathbf{x} = \mathbf{b}$ y después vemos el error producido, con la norma 2 ($\|e\|_2 \equiv \sqrt{\sum_{i=1}^{N} |S_i - x_i|^2} = \sqrt{(\mathbf{S}-\mathbf{x})^H(\mathbf{S}-\mathbf{x})}$). Así podremos comprobar que la solución del sistema es correcta.

Pero los resultados interesantes en este punto, son los de escalado con el número de procesos, ya que, el programa demostró ser capaz de resolver matrices que ocupan una mayor cantidad de memoria de la que cualquier nodo posee. Como adelantamos en la introducción, el parámetro que nos aportará la información queda definido por la ley de Amdahl:

$$G = \frac{T_1}{T_N} \tag{4.1}$$

Siendo $T_1$ el tiempo de ejecución con un sólo proceso y $T_N$ con $N$ procesos. Este parámetro representa la ganancia de la velocidad de ejecución de un programa en paralelo, *speedup* en terminología anglosajona.

La aplicación práctica es trivial, ejecutamos el programa en cuestión con un proceso, obteniendo así el numerador, y con $N$ procesos, obteniendo el denominador.

En las figuras 4.8-4.17 encontramos el factor de *speedup*, para las diferentes matrices, agrupado por el número de procesos por nodo 1, 2, 4 y 8, como podemos ver en la leyenda. Todos los resultados presentados corresponden a una segunda batería de pruebas, pues debido al mal funcionamiento de uno de los nodos, tuvimos que volver a realizar todas las simulaciones.

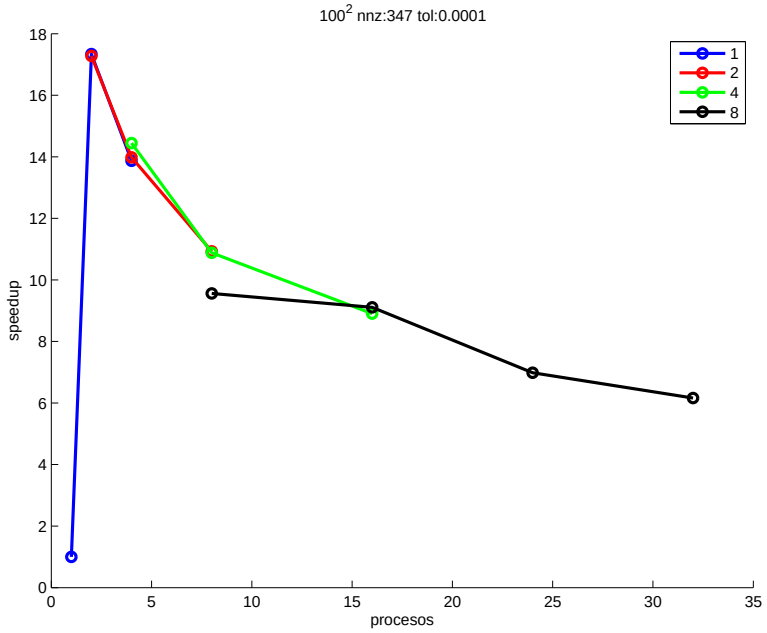
Podemos ver que para las dos primeras matrices domina la variabilidad de las iteraciones, pues llegan a crecer un 200 % con el número de procesos. La cuarta matriz, no consigue una curva deseable, porque pese a tener más de cien mil incógnitas, se resuelve en apenas diez segundos, y esto produce que los efectos secuenciales de carga de la matriz se vean claramente reflejados. La última matriz, funciona peor de lo esperado, pues podemos comprobar que en último punto, correspondiente a 32 procesos, tarda mucho más de lo esperado; no obstante podemos comprobar que cuando incrementamos la carga paralela (incrementando el número de iteraciones necesarias para llegar a la solución) el *speedup* se incrementa.

Para hacernos una idea de los tiempos de ejecución y de la variación de las iteraciones listamos un par de ejecuciones de cada matriz en la siguiente tabla:

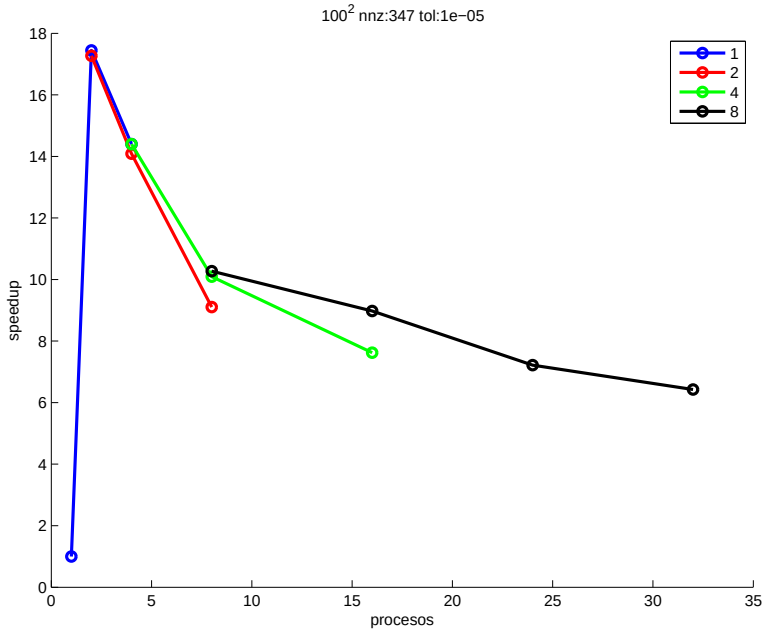
| **Matriz** | **Procesos** | **Tol** | **Iteraciones** | **Tiempo** |
|---|---|---|---|---|
| Nos4 | 2/1 | $1e^{-4}$ | 29 | 0,3 |
| Nos4 | 32/8 | $1e^{-7}$ | 118 | 0,880 |
| Nos3 | 24/8 | $1e^{-7}$ | 1023 | 1,892 |
| Nos3 | 32/8 | $1e^{-7}$ | 697 | 1,679 |
| Bloweybq | 16/4 | $1e^{-6}$ | 1115 | 4,802 |
| Bloweybq | 16/8 | $1e^{-6}$ | 1115 | 4,903 |
| 2cubes_sphere | 1/1 | $1e^{-7}$ | 10 | 9,293 |
| 2cubes_sphere | 32/8 | $1e^{-7}$ | 17 | 7,625 |
| Ecology2 | 1/1 | $1e^{-6}$ | 149 | 244,069 |
| Ecology2 | 32/8 | $1e^{-6}$ | 174 | 33,332 |

Los tiempos están en segundos, la columna procesos muestra $N/M$, siendo $N$ el número total de procesos y $M$ el número de ellos en cada nodo respectivamente. La tabla muestra como es la variación del número de iteraciones, con la matriz Nos4, por ejemplo, y cuantas operaciones se realizan para obtener la solución de 2cubes_sphere.

La conclusión que podemos extraer es que pese a lograr la convergencia y conseguir resolver problemas de mayor tamaño que la memoria física de un sólo nodo, la variabilidad de las iteraciones, siempre dependientes del número de procesos, no nos permiten calcular un valor de *speedup* válido.

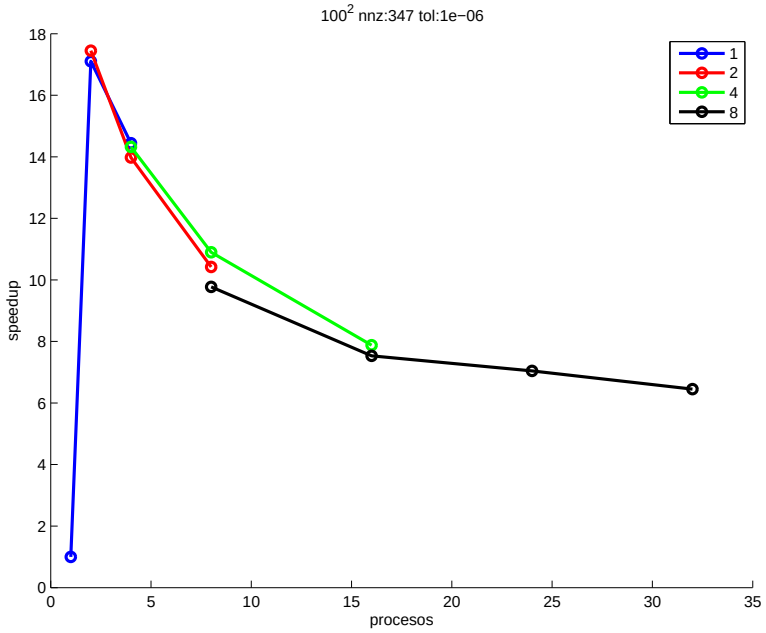



(a) Matriz Nos4 tol:0.0001

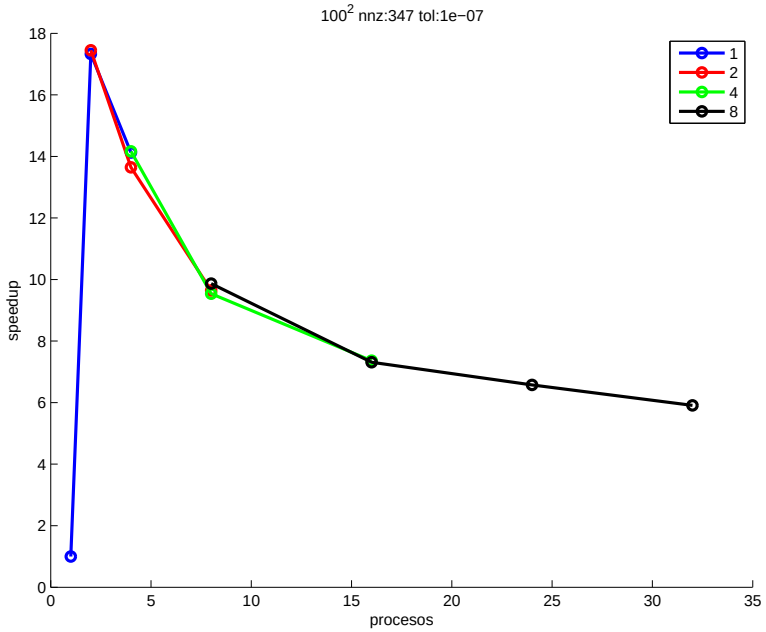



(b) Matriz Nos4 tol:1e-05

Figura 4.8: Resultados de PETSc I.

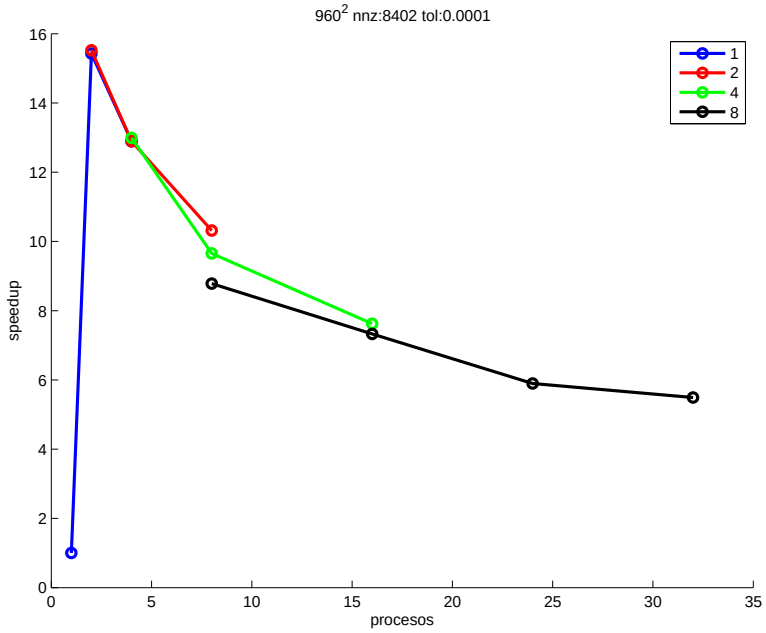



(a) Matriz Nos4 tol:1e-06

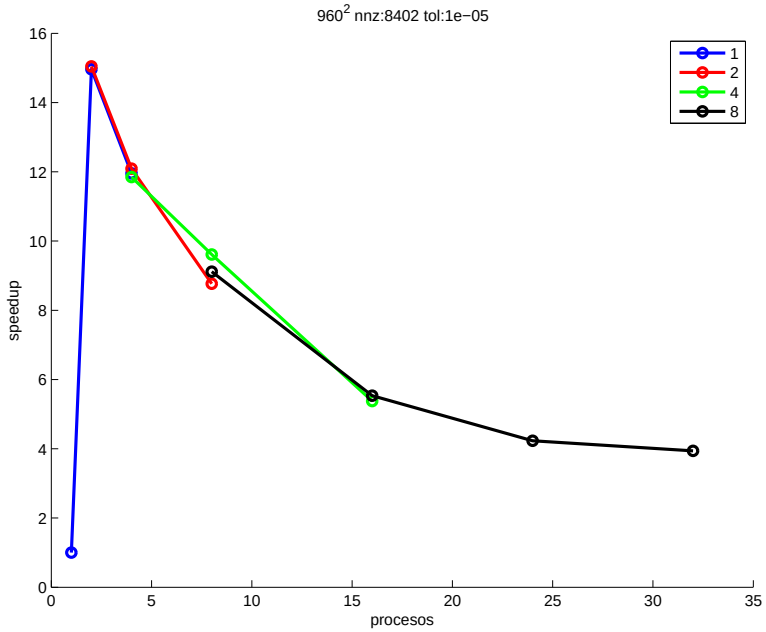



(b) Matriz Nos4 tol:1e-07

Figura 4.9: Resultados de PETSc II.

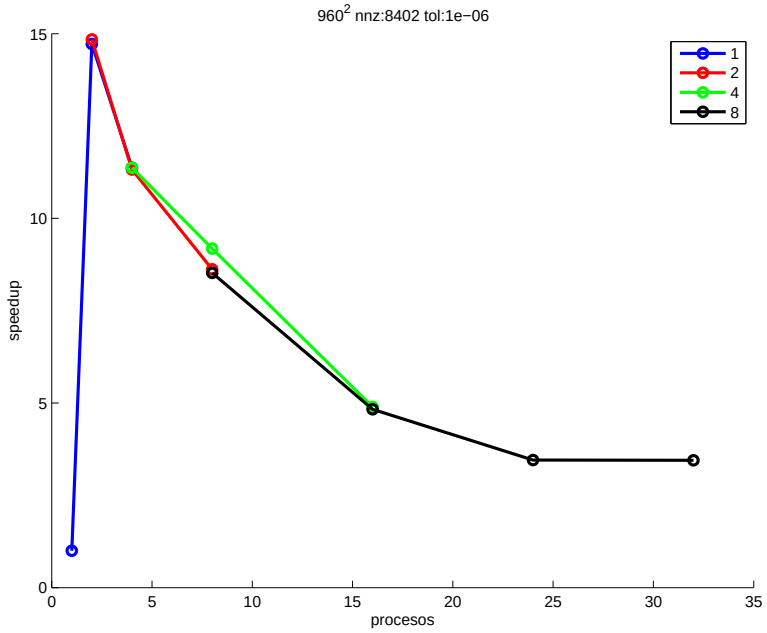



(a) Matriz Nos3 tol:0.0001

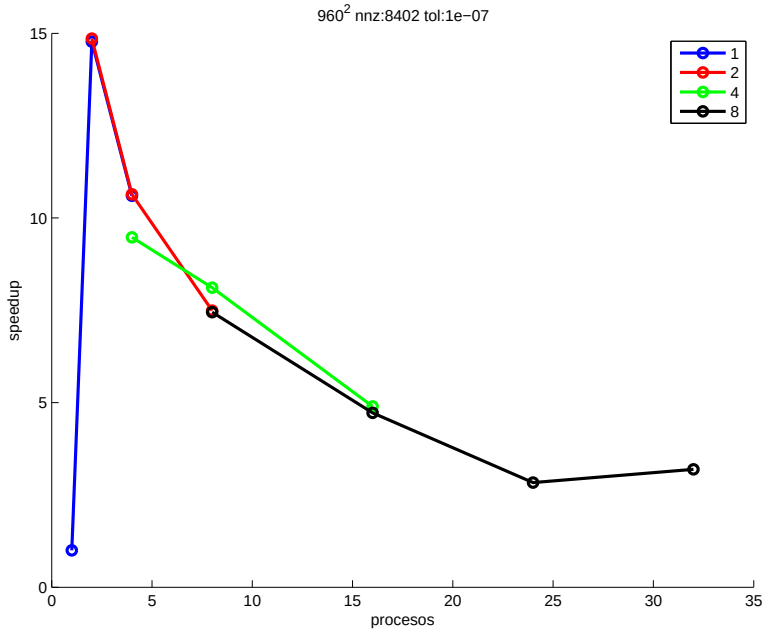



(b) Matriz Nos3 tol:1e-05

Figura 4.10: Resultados de PETSc III.

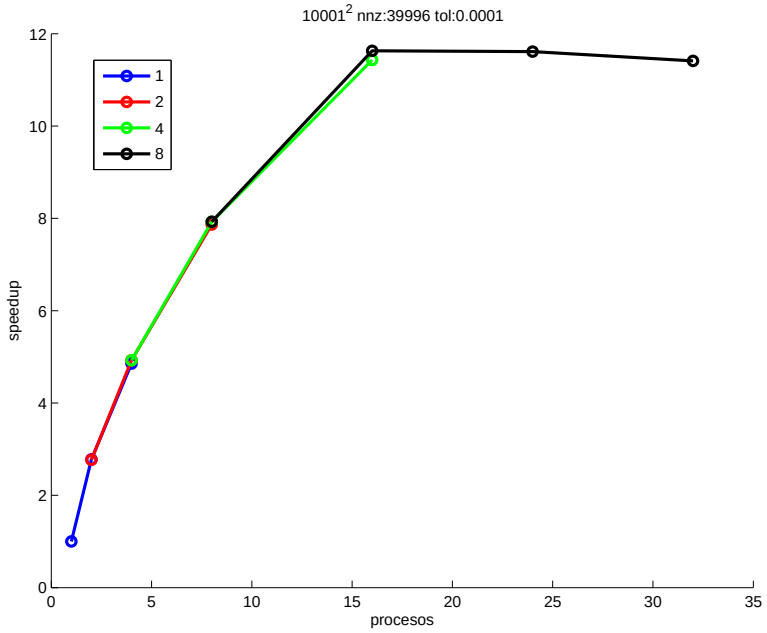



(a) Matriz Nos3 tol:1e-06

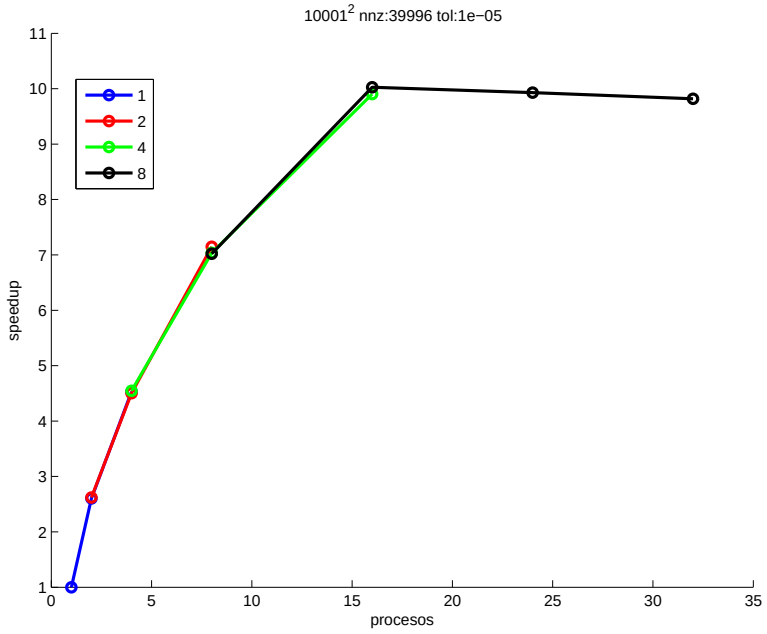



(b) Matriz Nos3 tol:1e-07

Figura 4.11: Resultados de PETSc IV.

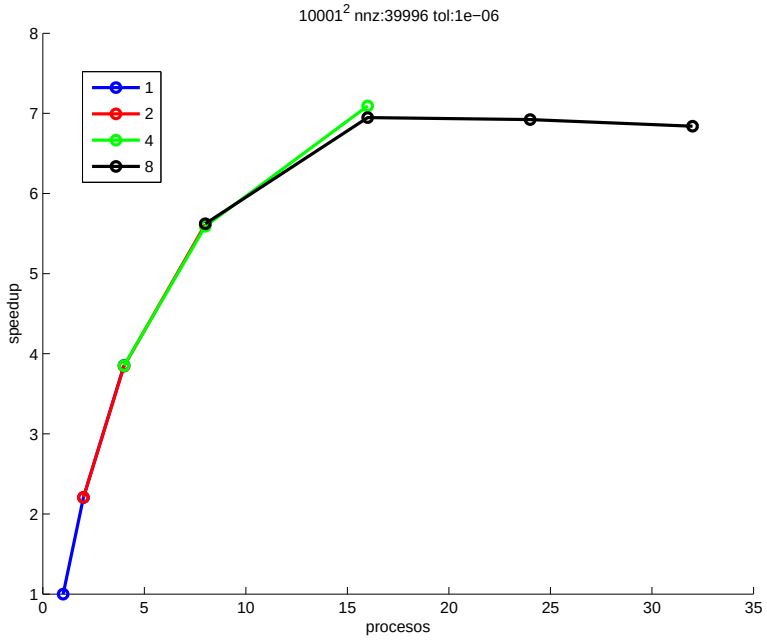



(a) Matriz Bloweybq tol:0.0001

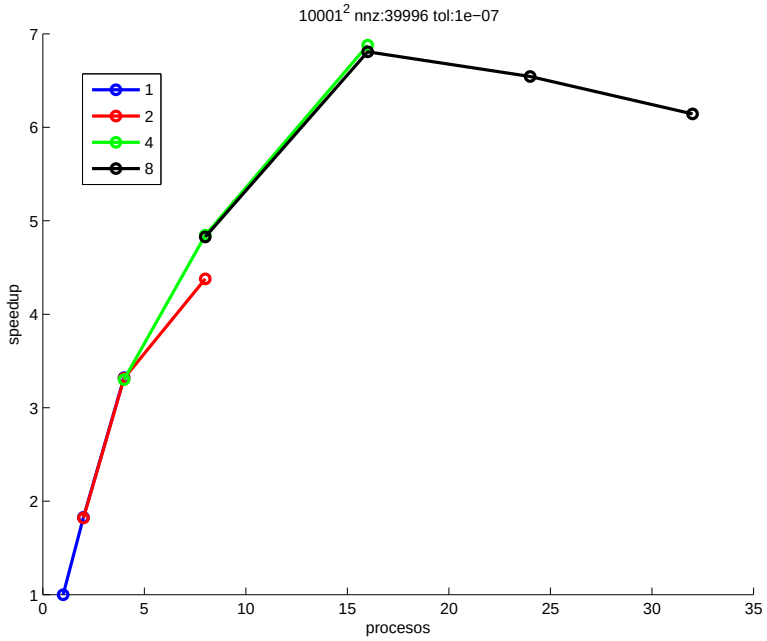



(b) Matriz Bloweybq tol:1e-05

Figura 4.12: Resultados de PETSc V.

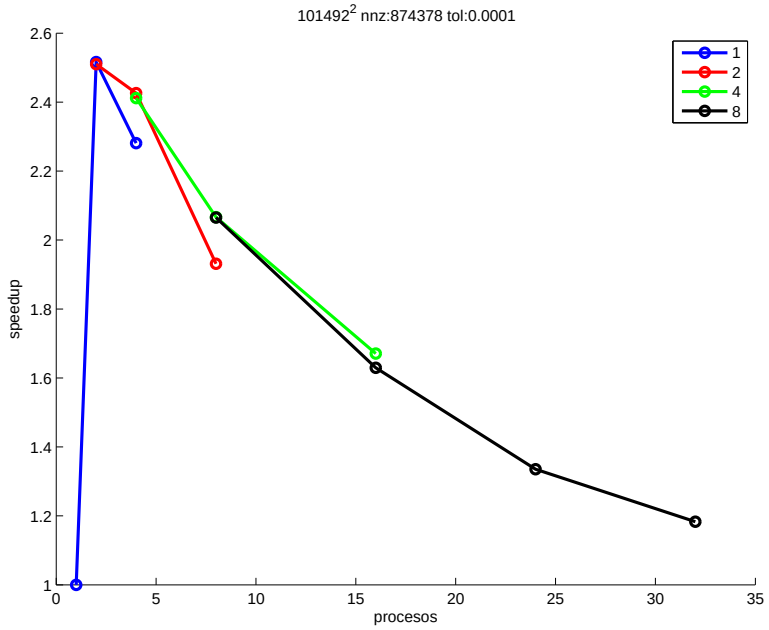



(a) Matriz Bloweybq tol:1e-06

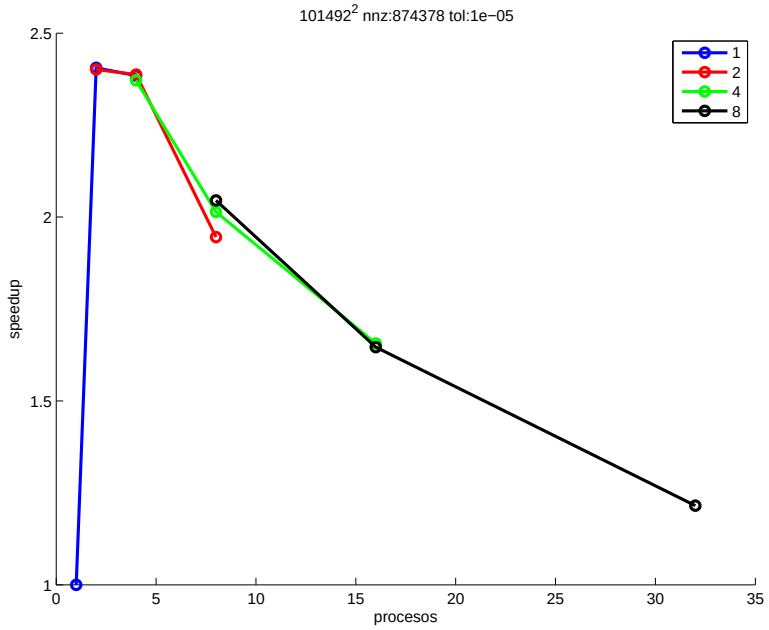



(b) Matriz Bloweybq tol:1e-07

Figura 4.13: Resultados de PETSc VI.

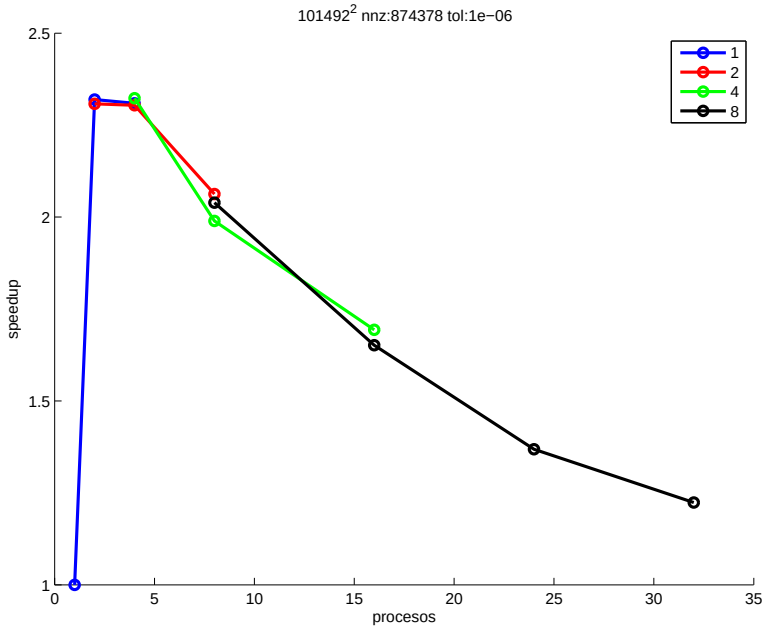



(a) Matriz 2cubes_sphere tol:0.0001

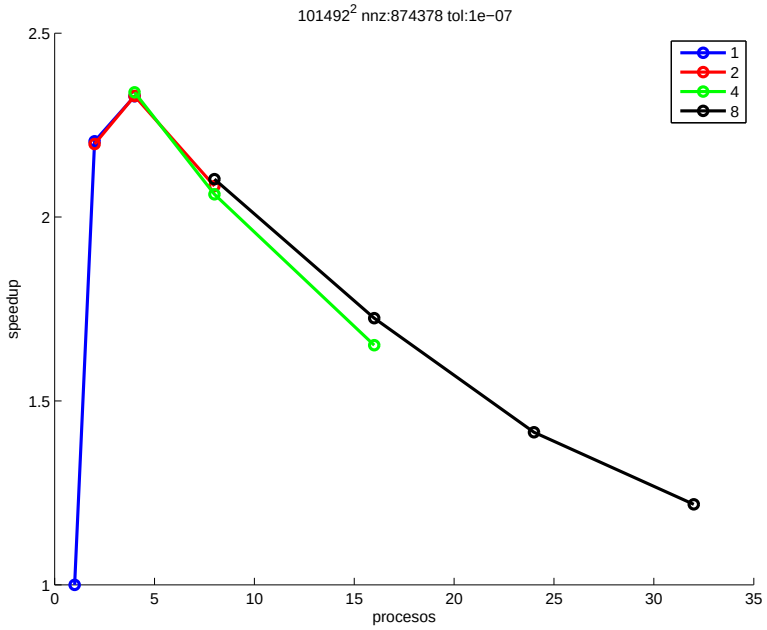



(b) Matriz 2cubes_sphere tol:1e-05

Figura 4.14: Resultados de PETSc VII.

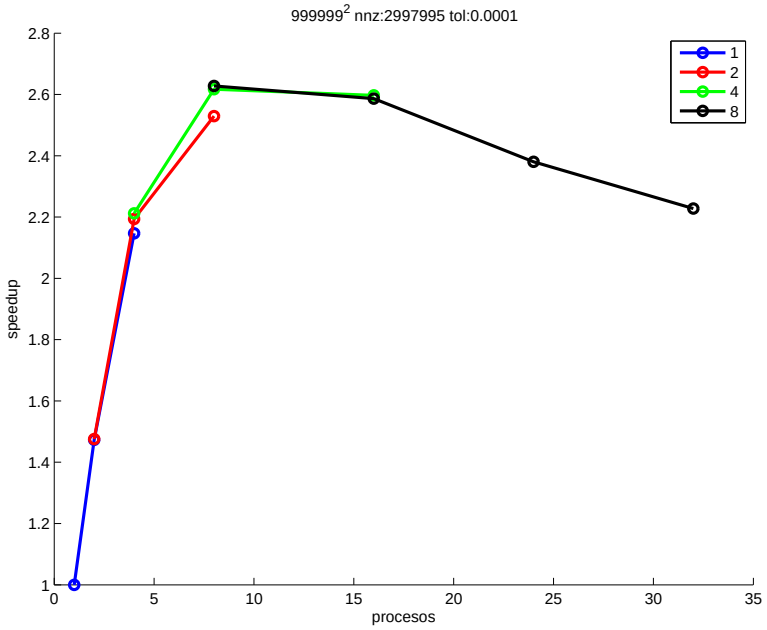



(a) Matriz 2cubes_sphere tol:1e-06

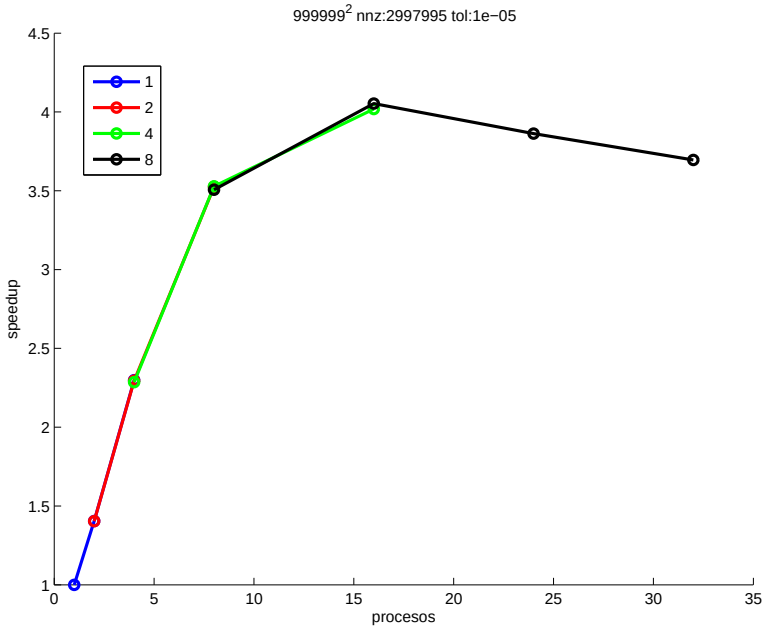



(b) Matriz 2cubes_sphere tol:1e-07

Figura 4.15: Resultados de PETSc VIII.




(a) Matriz Ecology2 tol:0.0001




(b) Matriz Ecology2 tol:1e-05

Figura 4.16: Resultados de PETSc XI.

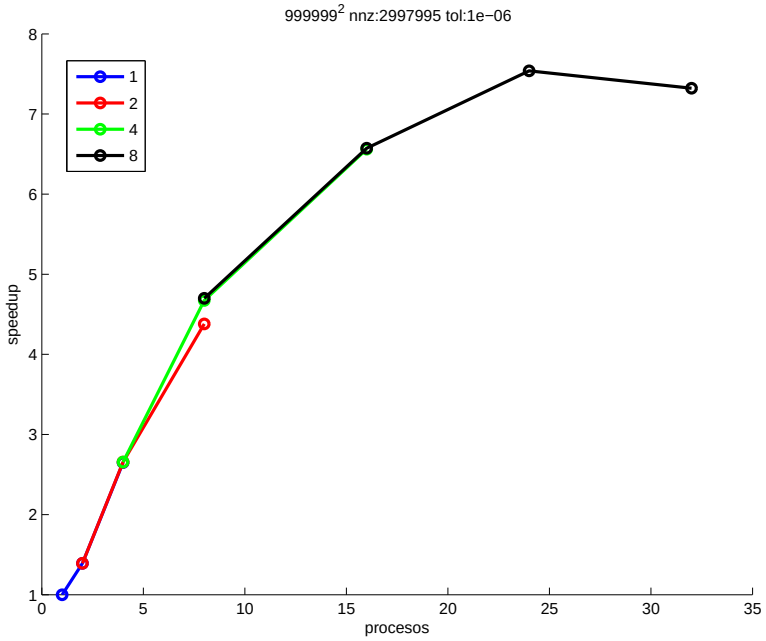



Figura 4.17: Resultados de PETSc X.

## 4.3. Test de un resolvedor directo multifrontal: HibridoFEM3D

El código denominado HibridoFEM3D utiliza en este caso un resolvedor directo de tipo multifrontal para matrices dispersas. Puede utilizar HSL, MUMPS y *PETSc* como vimos en 4.6, en este caso utilizaremos MUMPS. Los métodos directos se basan en una eliminación de Gauss, sabiendo como es el proceso de ensamblado. Dicho de otra forma, para ensamblar la matriz $[A]$, necesitamos sumar las matrices de los elementos que describen el dominio problema y en lugar de ensamblar la matriz podemos calcular, por eliminación gaussiana, la factorización de la matriz $[A]$, sin ensamblarla. Simplemente coge un subconjunto de elementos, los ensambla en una pequeña matriz densa (conocida como frente), y los procesa siguiendo con las relaciones entre los elementos (relaciones de ensamblado). Al final consigue una descomposición de tipo LU o Choleski de la matriz, sin ensamblar la matriz.

Con este simulador de problemas de radiación y dispersión, que hace uso de MUMPS para la factorización, simulamos una esfera dieléctrica, que podemos ver representada en la figura 4.18.

Lo interesante de este método es la posibilidad de hacer uso extensivo de BLAS a nivel 3, por lo que puede experimentar una mejora del rendimiento dejando cores

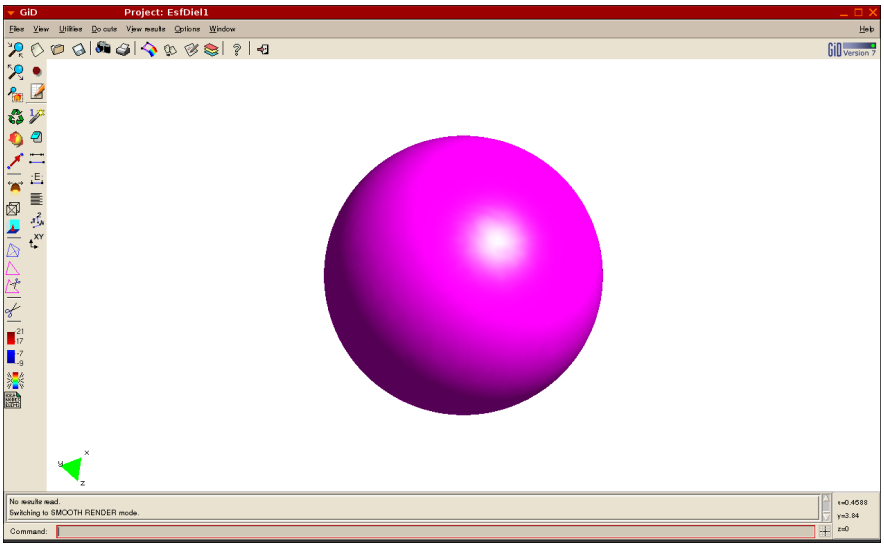



Figura 4.18: Esfera dieléctrica resuelta con HibridoFEM3D.

libres en los nodos. Esto es debido a las optimizaciones a nivel de compilador. Estas optimizaciones se traducen en un incremento del número de hilos que tiene la aplicación, realizando pequeñas paralelizaciones de código[4]. Trabajos anteriores del grupo indican diferencias muy significativas en el uso o no del tercer nivel de BLAS (concretamente en HSL secuencial)

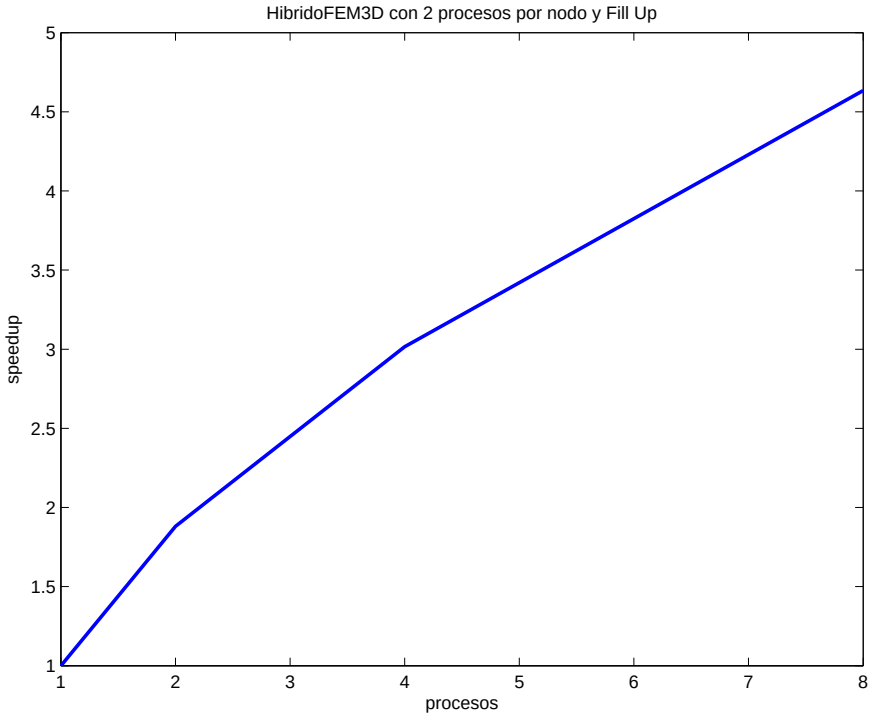
Para que los resultados sean comparables, el simulador fue puesto en modo *in core*, que es el modo de operación que usa solamente la memoria RAM del equipo donde se ejecuta, frente al modo *out of core* donde se usa también el disco duro.

Los resultados muestran el tiempo de factorización de la matriz, por lo que de existir una variación significativa se verá claramente reflejada. Inicialmente utilizamos una ordenación *fill up* (rellenando hasta la capacidad máxima cada nodo antes de pasar al siguiente).

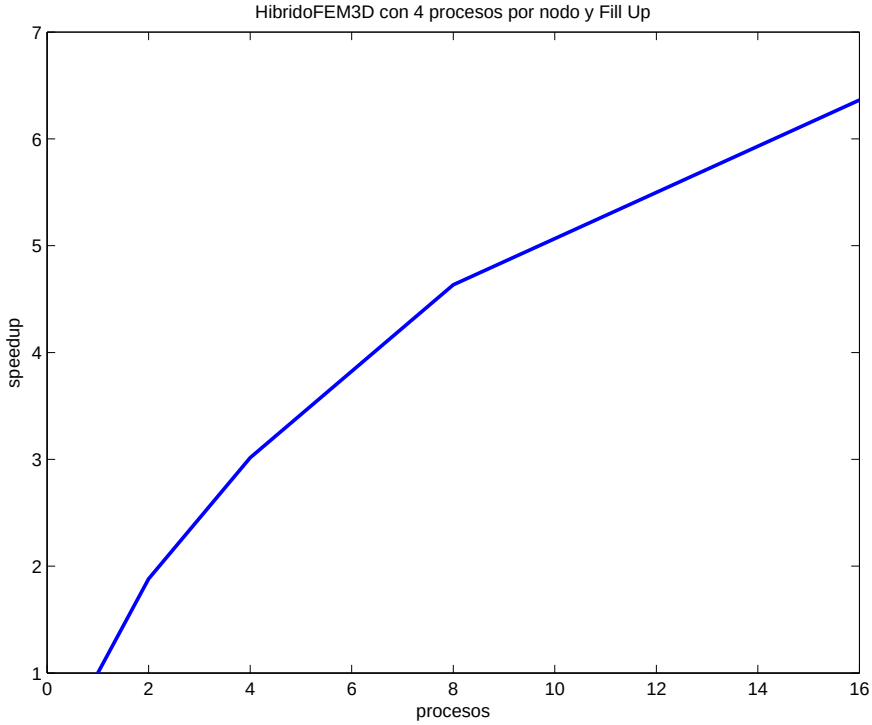
Cambiando la política de distribución de los procesos a *round robin* que distribuye uniformemente los procesos por el cluster, obtenemos los resultados que mostramos en la figura 4.3.

Si comparamos las gráficas, podemos ver que el factor *speedup* no varía en el primer grupo, con estrategia *fill up*. Comparando éstas con la última, con política *round robin*, podemos apreciar cambios leves en el factor *speedup*, es decir, no apreciamos el paralelismo a nivel de hilo (*thread*).

[4]Es gracias a que BLAS en su tercer nivel incluye directivas OpenMP

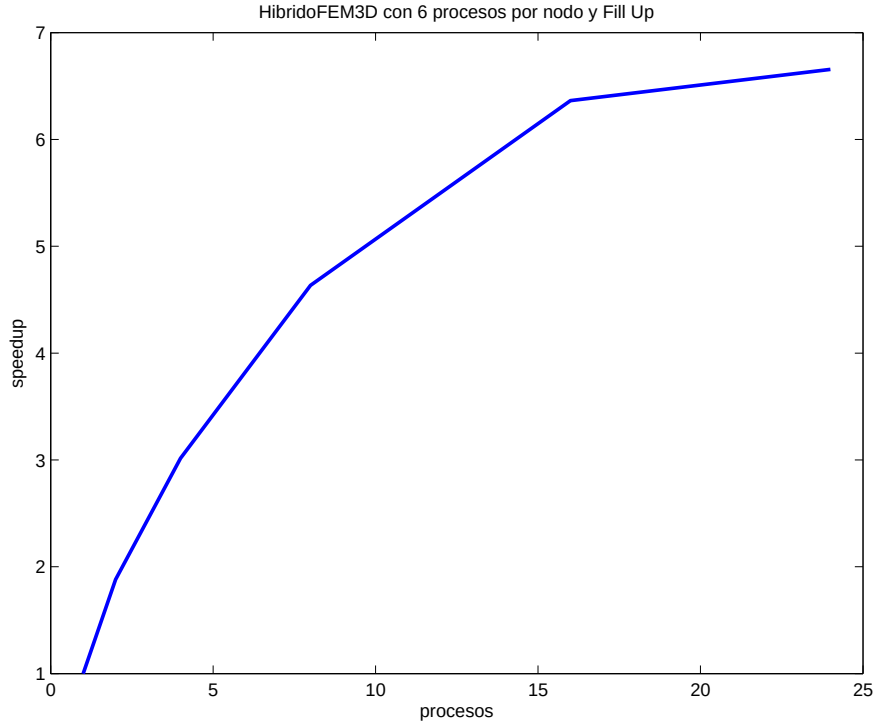



(a) Dos procesos por nodo.

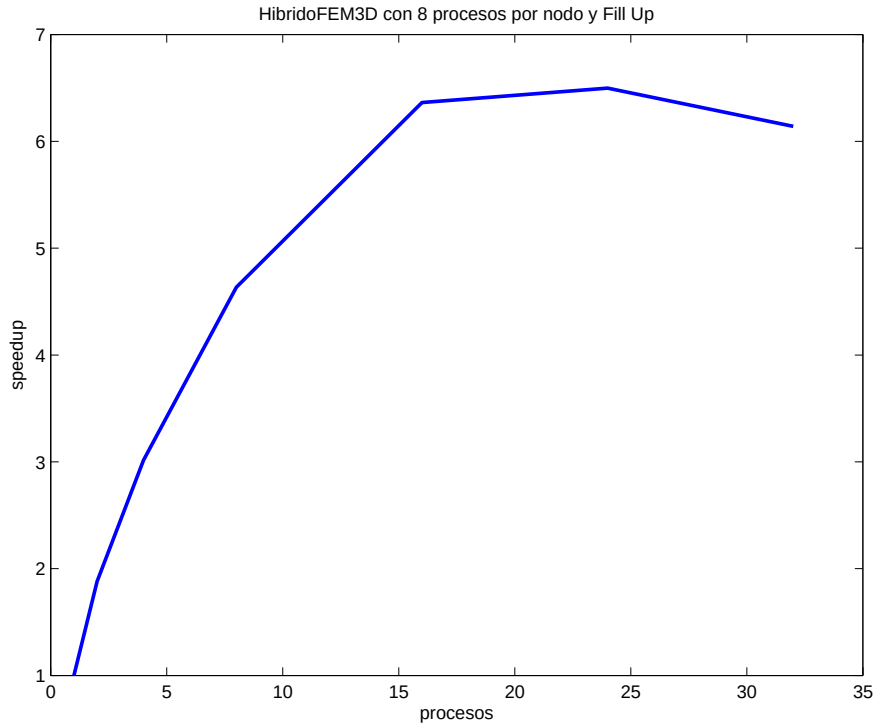



(b) Cuatro procesos por nodo.

Figura 4.19: Test de HibridoFEM3D I.




(a) Seis procesos por nodo.




(b) Ocho procesos por nodo.

Figura 4.20: Test de HibridoFEM3D II.

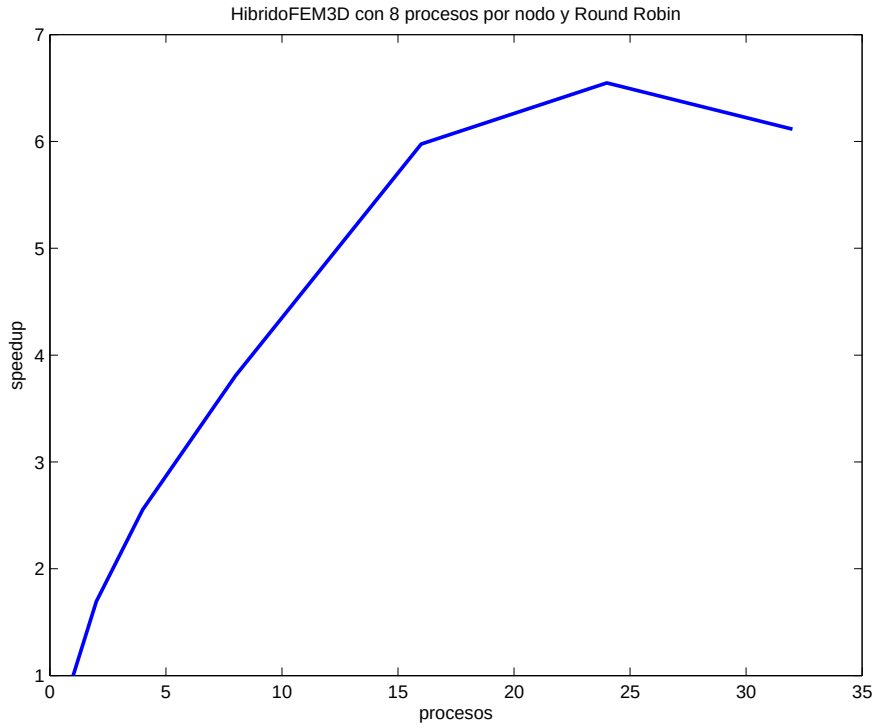



Figura 4.21: Test de HibridoFEM3D III.

## 4.4. Resolvedor de matrices densas: TIDES

TIDES ([31] y [36]) es un simulador de problemas de radiación y dispersión basado en MoM; su resolvedor trata matrices densas paralelizando el cálculo con ScaLAPACK [37] o PLAPACK [38].

ScaLAPACK es una librería algebraica paralela pensada para ser empleada bajo paradigmas de memoria distribuida, siendo recomendado su uso con MPI. Además trata eficazmente tanto las matrices densas, como las matrices dispersas.

PLAPACK es una librería algebraica paralela basada en MPI, que paraleliza las matrices y ofrece un interfaz amigable de cara al usuario.

TIDES tiene la opción de resolver tanto *in core*, como *out of core*, con escasa diferencia en el rendimiento. Además TIDES utiliza funciones base de Rao-Wilton-Glisson (RWG) para geometrías simples y funciones base de orden superior para geometrías complejas. El uso de las funciones de base de orden superior reduce el tiempo de ensamblado de la matriz, aprovechando su estructura para balancear la carga entre los nodos participantes.

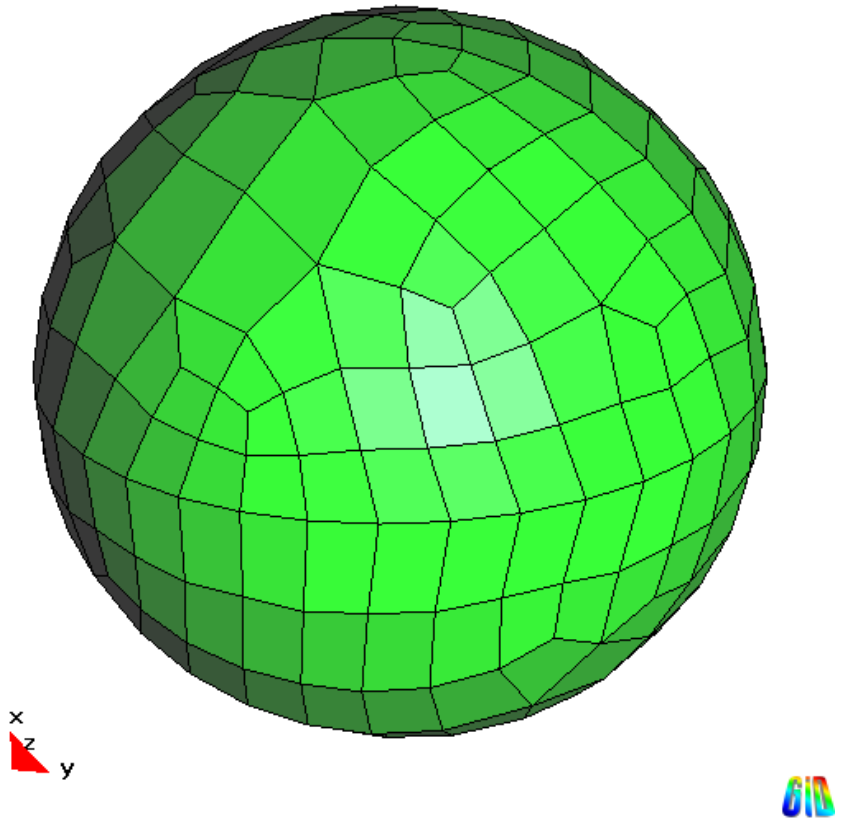



Figura 4.22: Esfera conductora resuelta con TIDES.

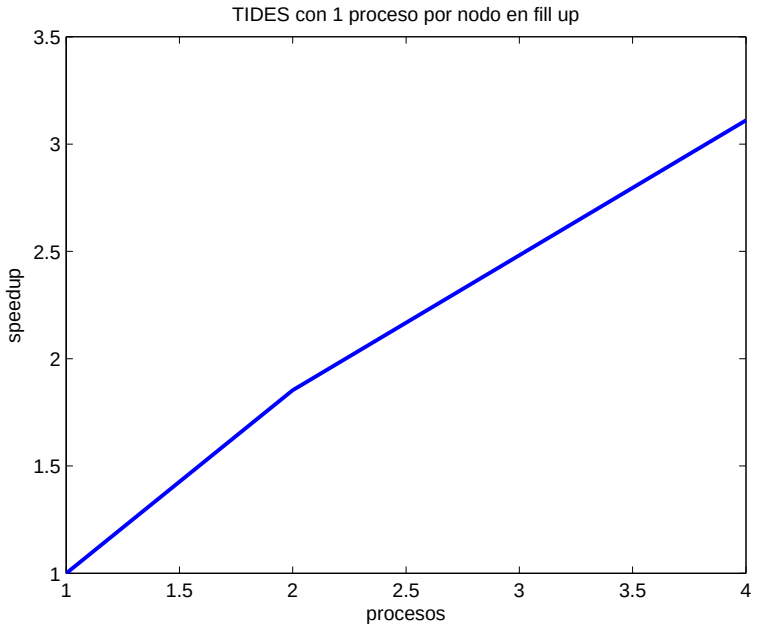
Debemos hacer hincapié en que TIDES es de los únicos resolvedores capaces de resolver por el Método de los Momentos en paralelo de forma *out of core*, pese a que en el presente proyecto sólo empleamos su implementación *in core*.

En este caso se simuló una esfera conductora que podemos ver en la figura 4.22.

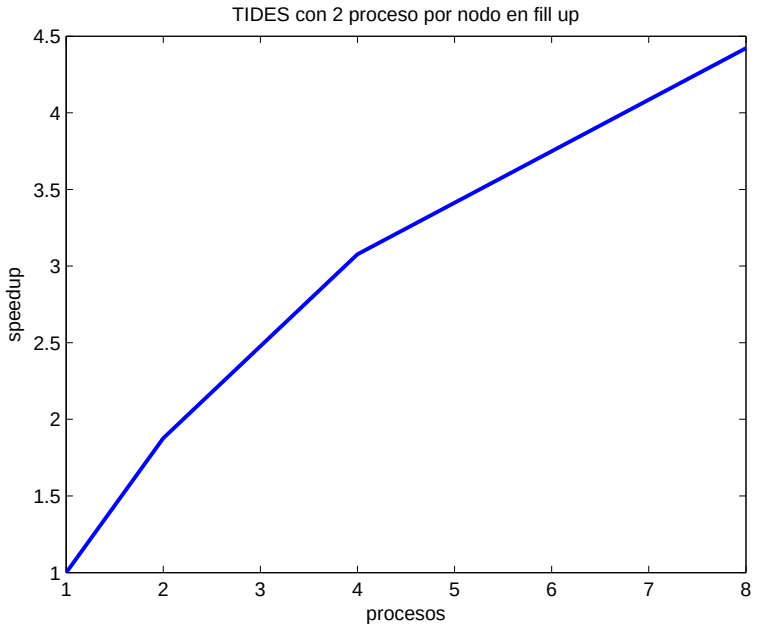
Nuevamente realizamos una serie de test con distinto número de procesos por nodo y con dos políticas de ordenamiento de procesos: *fill up* y *round robin*.

Las gráficas 4.23 y 4.24 representan el *speedup*, para 1, 2, 4 y 8 procesos por nodo y estrategia *fill up*. Las gráficas 4.25 y 4.26 lo hacen para *round robin*.

Resulta interesante comprobar que prácticamente no interviene el número de procesos por nodo. Esto significa que aprovecha correctamente la arquitectura multicore.

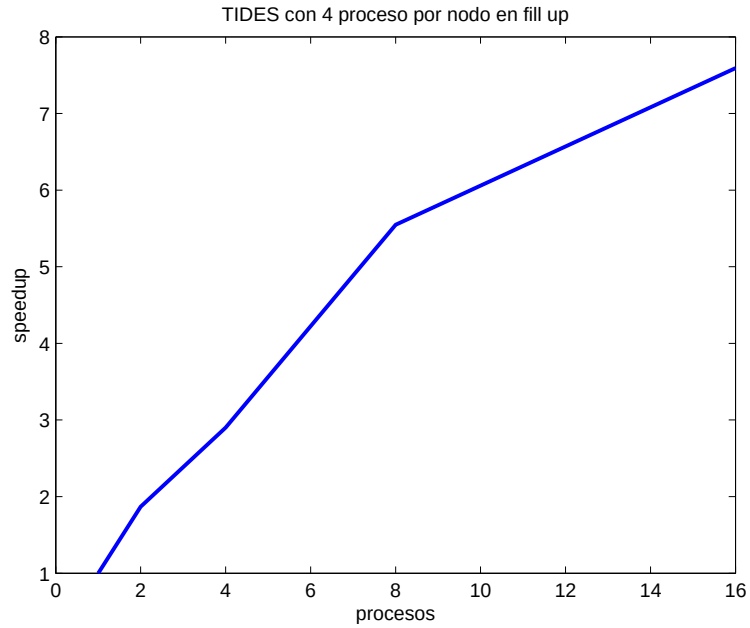



(a) Uno procesos por nodo.

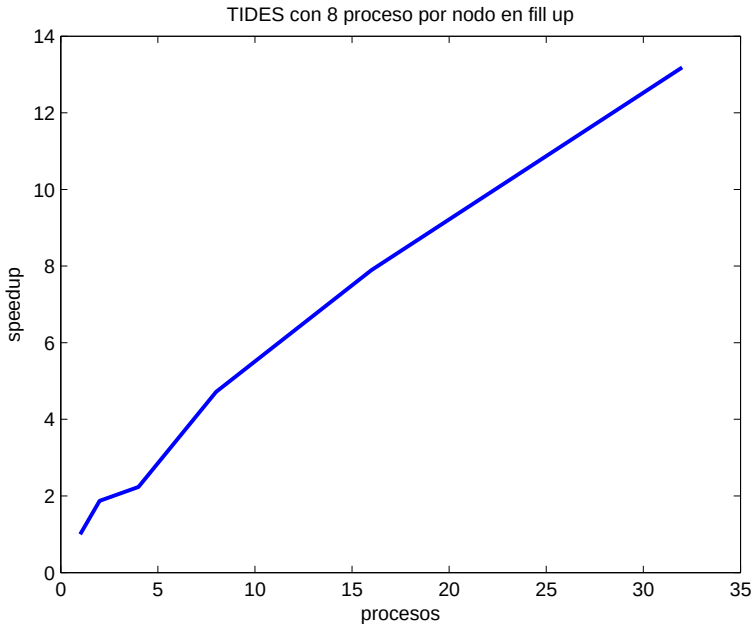



(b) Dos procesos por nodo.

Figura 4.23: Test de TIDES I.

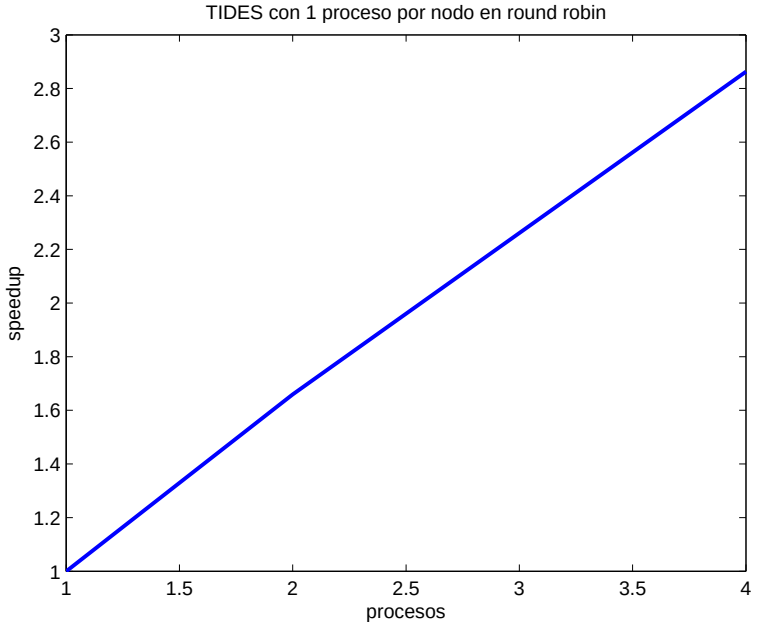



(a) Cuatro procesos por nodo.

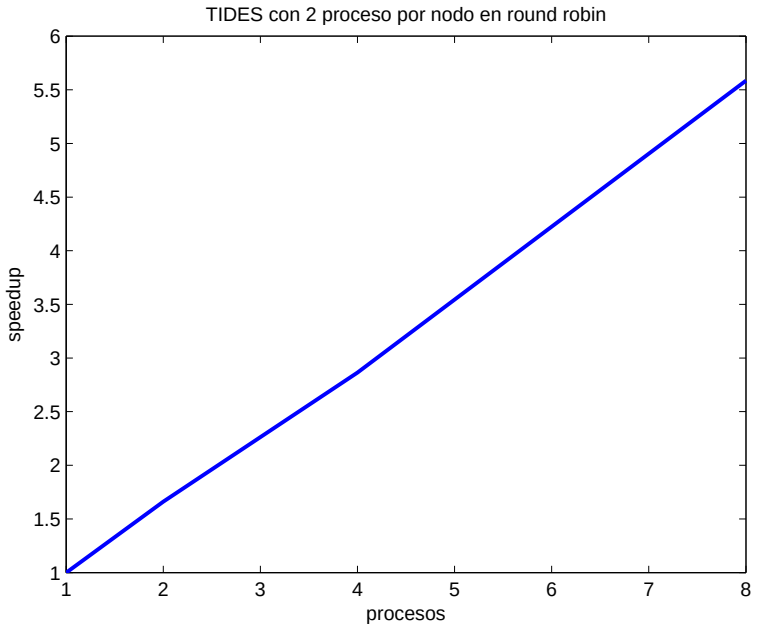



(b) Ocho procesos por nodo.

Figura 4.24: Test de TIDES II.

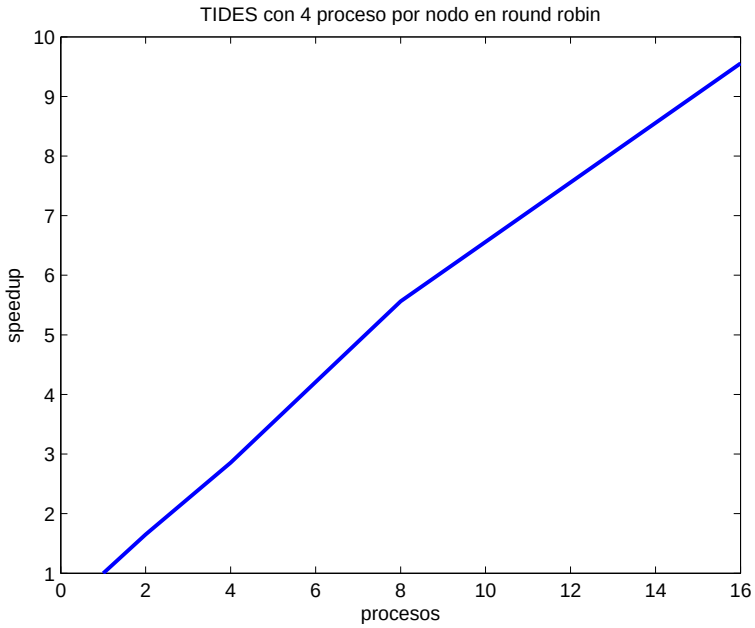



(a) Uno procesos por nodo.

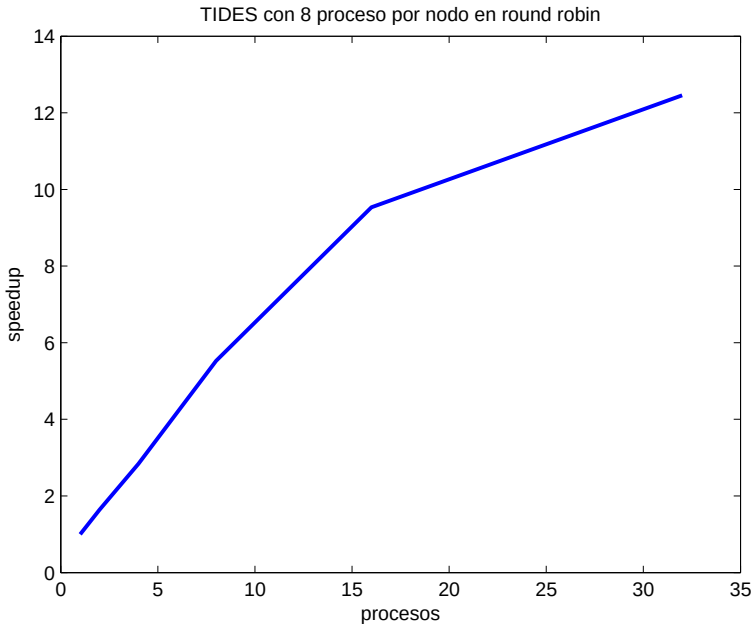



(b) Dos procesos por nodo.

Figura 4.25: Test de TIDES III.




(a) Cuatro procesos por nodo.




(b) Ocho procesos por nodo.

Figura 4.26: Test de TIDES IV.

## 4.5. Conclusiones

El funcionamiento del cluster, analizado con Linpack, ha resultado ser excelente, obteniendo 480 mil millones de operaciones por segundo.

Las pruebas de los diferentes algoritmos, nos han servido para comprobar el funcionamiento del cluster ante situaciones reales, y pese a que no todos los algoritmos se comportan del mismo modo. En conjunto, estamos satisfechos con los resultados obtenidos.

El desarrollo del resolvedor con *PETSc* resultó ser complicado y nos llevó más tiempo del desable. Aún así logramos resolver problemas "grandes"

# Capítulo 5

# Conclusiones y Futuras Líneas de Trabajo

## 5.1. Conclusiones

En el presente Proyecto Fin de Carrera se ha desarrollado un cluster para cálculo científico basado en Rocks [7] que permite la ejecución de tareas paralelas, reduciendo el tiempo de ejecución de las tareas y permitiendo el acceso a una mayor cantidad de memoria.

Se configuraron las instalaciones para una gran cantidad de software, así como servicios, que facilitan el uso, gestión y mantenimiento del cluster, simplificando las tareas administrativas.

Se comprobó el funcionamiento del sistema en cada paso, primero la comunicación con MPI y después la respuesta del conjunto de máquinas con *Linpack*, para pasar a comprobar el funcionamiento con diversos métodos de resolución. Todas las pruebas realizadas fueron exhaustivas y precisas, pues se controlaron todas las posibles variables con el fin de obtener resultados fiables.

Realizamos varias pruebas con problemas reales, incluyendo matrices dispersas y densas. Dentro de estas pruebas se codificó un resolvedor iterativo paralelo que, pese a no funcionar como esperábamos, cumplió con los requisitos.

Actualmente el cluster se encuentra completamente operativo y estable, demostrando ser una herramienta potente y flexible. Pese a que el posible campo de aplicación es muy reducido, cada vez más y más personas optan por este tipo de soluciones.

## 5.2. Futuras líneas de trabajo

El cluster ha cumplido con creces las expectativas que nos planteamos al inicio del proyecto, por lo que se plantea como puntos a tener en cuenta en la futura línea de trabajo:

- Actualización a Rocks 5.2, siempre una actualización completa del sistema es muy compleja aunque conveniente, además de resultar especialmente beneficioso, ya que, la separación entre la capa de *middleware* y la del sistema operativo de cada equipo se hace más fuerte.
- Mejora del sistema de backup de estado y desarrollo de Rolls para el software añadido.
- Añadir una segunda interfaz de red como red de cómputo de los nodos.

Con respecto al resolvedor iterativo implementado en este proyecto se plantean:

- Implementación de modelos híbridos MPI-OpenMP para mejorar el aprovechamiento de los procesadores.
- Implementación de un programa completo en *PETSc*.

# Apéndice A

# Introducción a los métodos de resolución de sistemas algrebraicos de ecuaciones

## A.1. Introducción

Para introducir los métodos numéricos vamos a partir de un sistema de ecuaciones $[A]\mathbf{x} = \mathbf{b}$:

$$\begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{bmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_N \end{pmatrix} \tag{A.1}$$

El tamaño de la matriz puede ser lo suficientemente grande como para consumir toda la memoria del equipo, por lo que es muy importante conocer como es la matriz, para, en la medida de lo posible, almacenar solamente la información imprescindible.

Por ejemplo, si la matriz es simétrica se almacenará la mitad superior (o inferior) de la matriz. Y si además resulta que la matriz es dispersa[1], que contienen gran cantidad de elementos iguales a cero, podemos suponer una forma de "banda"[2], y sólo se amacenan los elementos distintos de cero.

[1] En la bibliografía anglosajona se denominan *sparce matrix*

[2] Aunque no tenga forma de "banda", existen reordenamientos que la llevan a esta forma.

En el presente proyecto trataremos con matrices dispersas y densas, provenientes de la aplicación del Método de Elementos Finitos y del Método de los Momentos respectivamente.

## A.2. Métodos Directos

Intentemos resolver el sistema de ecuaciones, comenzando por un método directo; por ejemplo, tratemos de resolver un sistema de 3 ecuaciones con 3 incógnitas por el método de Gauss:

$$\begin{aligned} x_1 + 2x_2 &= 0 \\ 2x_1 + 5x_2 - x_3 &= 1 \\ 4x_1 + 10x_2 - x_3 &= -1 \end{aligned} \tag{A.2}$$

Ahora procedemos a triangularizar el sistema, restando la primera ecuación a la segunda ecuación dos veces y cuatro veces a la tercera:

$$\begin{aligned} x_1 + 2x_2 &= 0 \\ x_2 - x_3 &= 1 \\ 2x_2 - x_3 &= -1 \end{aligned} \tag{A.3}$$

Y ahora restamos a la tercera ecuación la segunda dos veces:

$$\begin{aligned} x_1 + 2x_2 &= 0 \\ x_2 - x_3 &= 1 \\ x_3 &= -3 \end{aligned} \tag{A.4}$$

Acabamos de triangularizar el sistema llevándolo de $[A]\mathbf{x} = \mathbf{b}$ a $[U]\mathbf{x} = \mathbf{c}$, y ahora tenemos un sistema triangular superior. De esta manera conseguimos calcular $x_3 = -3$ y en cada sustitución conseguimos otra incógnita. Realmente al llevar al sistema de ecuaciones a una forma triangular, lo que hacemos es factorizarlo, consiguiendo simplificar el sistema.

El coste computacional de hallar la solución ha sido de $O(N^3)$ para factorizar el sistema y $O(N^2)$ para las sustituciones. Al resolver el sistema empleamos $\frac{N^3}{3}$ multiplicaciones para calcular la triangularización, dado que el tiempo empleado en las sumas es despreciable frente a la multiplicación y que la sustitución al ser de inferior orden también es despreciable.

Si en lugar de proceder con el método de Gauss lo hubiéramos hecho con la regla de Cramer el coste computacional hubiera sido de $O(N!)$. Por lo que preferiremos una factorización antes que usar la regla de Cramer.

Debemos fijarnos que no hemos calculado la inversa de $[A]$ en ningún momento. La pregunta es: ¿es realmente necesario? La respuesta es que no es necesario. Para calcular la inversa necesitaremos aproximadamente el mismo número de operaciones, pero el doble de memoria, y la solución es la misma. Aunque podemos pensar que si queremos resolver el sistema con otro $\mathbf{b}$ calcular la inversa nos podría ahorrar tiempo, la respuesta vuelve a ser negativa. Para resolver sistemas con varios $\mathbf{b}$ podemos extender el sistema y proseguir de la misma manera, es decir:

$$[A][\mathbf{x}_1 \cdots \mathbf{x}_N] = [\mathbf{b}_1 \cdots \mathbf{b}_N] \tag{A.5}$$

Por lo que calcular la inversa de la matriz con el fin de resolver el sistema de ecuaciones es muy raro, debido a que la inversa contiene más información de la necesaria para llegar a la solución. Generalmente se procede a factorizar la matriz, de la siguiente forma:

$$[L][U]\mathbf{x} = \mathbf{b} \tag{A.6}$$

Donde la matriz $[U]$, triangular superior, es el resultado de aplicar el método de Gauss a la matriz y $[L]$, triangular inferior, contiene la información de las permutaciones realizadas para obtener $[U]$, en el ejemplo anterior:

$$\begin{aligned} [U] &= \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \\ [L] &= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 2 & 1 \end{bmatrix} \end{aligned} \tag{A.7}$$

Podemos observar como $[L]$ contiene la infomación de las operaciones (sustracciones) para llegar a la $[U]$ del ejemplo anterior. Para un vector $\mathbf{b}$ cualquiera podemos precondicionarlo, es decir, podemos transformar el sistema $[A]\mathbf{x} = \mathbf{b}$ en $[U]\mathbf{x} = \mathbf{c}$ sin más que multiplicar por $[L]^{-1}$ en ambos lados.

$$[U]\mathbf{x} = [L]^{-1}\mathbf{b} \tag{A.8}$$

Pero como hemos dicho no hace falta calcular la inversa, es decir $[U]$ es conocida, la calculamos a la vez que $[L]$ y calcular $\mathbf{c} = [L]^{-1}\mathbf{b}$ es lo mismo que calcular $[L]\mathbf{c} = \mathbf{b}$, con la ventaja que es triangular inferior, por lo que requerirá $O(N^2)$ operaciones.

$$\begin{aligned} [L]\mathbf{c} &= \mathbf{b} \\ [U]\mathbf{x} &= \mathbf{c} \end{aligned} \tag{A.9}$$

Para el caso en el que las matrices sean dispersas, el coste computacional asociado a la factorización $LU$ es de $O(N^2D/2)$ donde $D$ es la mayor distancia desde la diagonal a un elemento no nulo (comunmente denominada *banda* de la matriz).

Los problemas electromágneticos terminan siendo un sistema de ecuaciones algebraico, que resolveremos con un ordenador. Pero la aritmética de los ordenadores es finita, y la precisión de la aproximación también, veamos como afectan los errores a estos sistemas.

## A.3. Errores en métodos numéricos

Consideremos primero las fuentes de error, para posteriormente valorar como afectan al sistema. La aritmética que comparten la mayor parte de los ordenadores es el estándar ANSI/IEEE Standard 754-1985 for Binary Floating Point Aritmetic. Podemos ver un ejemplo en la figura A.3.

El número más pequeño que se puede expresar es $2^{-52} = 2{,}220446049250313 \cdot 10^{-16}$. Y como vemos en la figura anterior, donde $f$ es la mantisa, $e$ es el exponente, y $s$ es el signo, este formato representa los números como $x = s(1+f)^e$. Destacar que existen números especiales para operandos no definidos, como $NaN = 0/0$ o $Inf = 1/0$, con los que podremos lanzar excepciones y tratarlas. Lo importante


Figura A.1: Representación numérica de doble precisión.

es que existe un error mínimo derivado de discretizar la recta de los números reales, que siendo optimistas es el error mínimo que no podremos evitar.

Valoremos ahora un sistema de ecuaciones genérico con errores:

$$([A] + [\triangle A])(\mathbf{x} + \triangle\mathbf{x}) = \mathbf{b} + \triangle\mathbf{b} \tag{A.10}$$

Generalmente, pequeños errores en $[A]$ generan errores en $\mathbf{x}$, estos errores pueden ser más o menos grandes dependendiendo de la distribución inicial de autovalores de la matriz, en general se verá amplificado para valores del número de condicionamiento mayores que la unidad.

A partir de los autovalores $\lambda$ se pueden definir:

- El radio espectral de la matriz como

$$\rho([A]) \equiv \max_{\lambda \in \sigma([A])} |\lambda| \tag{A.11}$$

- El número de condicionamiento de la matriz, denotado como $\kappa$, que corresponde a la raíz cuadrada del cociente entre los autovalores máximo y mínimo de la matriz $[A]^H[A]$:

$$\kappa(A) \equiv \|[A]\|\|[A]^{-1}\| = \sqrt{\frac{\lambda_{max}}{\lambda_{min}}} \tag{A.12}$$

El radio espectral nos da una medida del condicionamiento para métodos iterativos que serán abordados en la próxima sección.

La relación entre $\kappa(A)$ y el error no es trivial, y no la demostraremos aquí, podemos encontrar la demostración en [34]. Se relacionan como sigue:

$$\frac{\|\mathbf{e_k}\|}{\|\mathbf{e_0}\|} \leq \kappa(A)\|\mathbf{r_k}\|\|\mathbf{r_0}\| \tag{A.13}$$

donde $\mathbf{e_n} = \mathbf{x} - \mathbf{x^k}$ y $\mathbf{r_n} = \mathbf{b} - [A]\mathbf{x^k}$.

El número de condicionamiento $\kappa(A) \in (0, \infty)$ dependiendo de como se corten los hiperplanos. Así, un sistema con una base ortonormal tiene $\kappa(A) = 1$. Sí no se cortan se tiene un sistema singular y $\kappa(A) = \infty$. De este modo, $\kappa(A)$ nos da una medida de la calidad de la base del subespacio generado por la matriz. El número de condicionamiento, cuando es muy alto, se traduce en pérdida de dígios de precisión (basicamente 1 dígito por cada orden de magnitud) en los método directos. En el caso de métodos iterativos, objeto de la próxima sección, los problemas son de convergencia.

## A.4. Métodos iterativos

Si reescribimos el sistema de ecuaciones $[A]\mathbf{x} = \mathbf{b}$ de la siguiente manera:

$$[M]\mathbf{x} = ([M] - [A])\mathbf{x} + \mathbf{b} \tag{A.14}$$

Podemos iterar de la siguiente manera para obtener $\mathbf{x}$:

$$[M]\mathbf{x}^{k+1} = ([M] - [A])\mathbf{x}^k + \mathbf{b} \tag{A.15}$$

De este modo, podemos definir nuevamente el error como:

$$\mathbf{e}^{k+1} = [M]^{-1}([M] - [A])\mathbf{e}^k = [G]\mathbf{e}^k \tag{A.16}$$

Es fácil ver que al iterar, la matriz $[G]^k \to [0]$ si todos los autovalores son (en módulo) menores que la unidad. Ello ocurrirá si el radio espectral, $\rho(G)$, es menor que uno.

A continuación mostraremos algunos métodos iterativos (podemos verlos en [34] o [39]) para llegar a la solución de (A.15), y culminaremos explicando algunos de los precondicionadores más comunes, así como su utilidad.

Podemos definir a partir de la matriz $[A]$:

- $[D]$ como la diagonal de $[A]$
- $[-E]$ como la parte triangular inferior de $[A]$

- $[-F]$ como la parte triangular superior de $[A]$

Entonces la iteración por el método de Jacobi viene expresada cambiando en la equación (A.15) $[M]$ por $[D]$, quedando:

$$[D]\mathbf{x}^{k+1} = ([E] + [F])\mathbf{x}^k + \mathbf{b} \tag{A.17}$$

Si además de la diagonal de la matriz tomamos la parte triangular inferior, sustituyendo $[M]$ por $[D - E]$, tenemos el método de Gauss-Seidel:

$$[D - E]\mathbf{x}^{k+1} = ([F])\mathbf{x}^k + \mathbf{b} \tag{A.18}$$

La ventaja del método de Jacobi es que el sistema de ecuaciones a resolvr en cada caso tiene una matriz diagonal que es trivial de invertir (y trivial de paralelizar). En el caso de Gauss-Seidel, la matriz es de tipo triangular inferior, que comienza a ser algo más costosa de invertir (igualmente, requiere más comunicaciones su paralelización). Podemos ver las las ecuaciones anteriores ((A.17) y (A.18)) siguen la forma $[M]\mathbf{x}_{k+1} = [N]\mathbf{x}_k + \mathbf{b}$, podemos definir hacer $[M] = w^{-1}[D] - [E]$, donde el parámetro $w$ toma valores en el intervalo $(0, 2)$. Si $w$ es mayor que la unidad el método se llama de sobrerelajación y en caso contrario de subrelajación[3] (notar que el valor $w = 1$ es el método de Gauss-Seidel). Las iteraciones quedarían definidas por:

$$[[D] - w[E]]\mathbf{x}^{k+1} = (w[F] + (1 - w)[D])\mathbf{x}^k + w\mathbf{b} \tag{A.19}$$

Que es el llamado método de SOR[4], que para matrices simétricas o hermíticas se define en dos pasos y se denomina SSOR[5]:

$$\begin{aligned} [M]_1 &= w^{-1}[D] - [E] \\ [M]_2 &= w^{-1}[D] - [F] \\ [[D] - w[E]]\mathbf{x}^{k+\frac{1}{2}} &= (w[F] + (1 - w)[D])\mathbf{x}^k + w\mathbf{b} \\ [[D] - w[F]]\mathbf{x}^{k+1} &= (w[E] + (1 - w)[D])\mathbf{x}^{k+\frac{1}{2}} + w\mathbf{b} \end{aligned} \tag{A.20}$$

Una vez explicados los métodos iterativos más básicos (de tipo estacionario), introduciremos los precondicionadores.

[3] En la literatura anglosajona se denotan como *overrelaxation* y *underrelaxation* respectivamente.

[4] Successive Overrelaxation Method

[5] Symetric SOR

### A.4.1. Precondicionadores

La idea es simple. Como se expresó con anterioridad, consiste en multiplicar el sistema de ecuaciones por una matriz, a ambos lados que transforma el sistema $[A]\mathbf{x} = \mathbf{b}$ en otro $[G]\mathbf{x} = \mathbf{f}$, de forma que la nueva matriz $[G]$ tenga unas propiedades espectrales determinadas (que impliquen una mejor convergencia).

Por otro lado, los métodos descritos en el apartado anterior, pueden reformularse como el uso de un precondicionador determinado. Considerando los siguientes precondicionadores: $[D]$, $[D-E]$ y $[[D]-w[E]]$, podemos reescribir, respectivamente, las equaciones (A.17), (A.18) y (A.19):

$$\mathbf{x}^{k+1} = [D]^{-1}([E]+[F])\mathbf{x}^k + [D]^{-1}\mathbf{b} \tag{A.21}$$

$$\mathbf{x}^{k+1} = [D-E]^{-1}([F])\mathbf{x}^k + [D-E]^{-1}\mathbf{b} \tag{A.22}$$

$$\mathbf{x}^{k+1} = [[D]-w[E]]^{-1}(w[F]+(1-w)[D])\mathbf{x}^k + [[D]-w[E]]^{-1}w\mathbf{b} \tag{A.23}$$

Para el método SSOR, no es tan sencillo, pero el precondicionador es

$$\frac{1}{2-w}[[D]-w[E]][D]^{-1}[[D]-w[F]]$$

.

## A.5. Métodos de Krylov

Dada una matriz,$[A]$, de $NxN$ y un vector,$\mathbf{b}$, $N$-dimensional, un subespacio de Krylov, es el espacio vectorial generado por imagenes de $\mathbf{b}$ en primeras $r$ potencias de $[A]$, esto es:

$$K_r([A],\mathbf{b}) = span\{[A]^k\mathbf{b} \quad \forall k \in \{0,\ldots,r-1\}\}$$

.

Ahora podemos entender que si proyectamos el sistema algebraico a este subespacio obtenemos métodos iterativos no estacionarios. Debemos mencionar que generalmente necesitamos matrices semidefinidas positivas.

Sin entrar en detalles presentaremos los dos métodos básicos utilizados en el proyecto:

- El método GMRES (*Generalizated Minimal RESidual*), consiste en minimizar la norma del residuo de la proyección, $\mathbf{r_o} = [A]\mathbf{x} - \mathbf{b}$, en un subespacio de Krylov ortogonalizado.
- CG (*Conjugated Gradient*), consite en proyecciones ortogonales en el subespacio de Krylov $K_m(\mathbf{r_0}, [A])$ con $\mathbf{r_0}$ el residuo inicial.

Existen multitud de variaciones de los dos métodos anteriores, como por ejemplo BiGCSTAB (*BiConjugated Gradient STABilized*). Los algoritmos especificos de cada uno, los podemos encontrar en [34].

# Apéndice B

# Guía de instalación

En el presente anexo pretendemos exponer los pasos necesarios para instalar Rocks 4.3. En las diferentes secciones veremos como se desarrolla la instalación tanto en el nodo maestro o *frontend*, como en los nodos.

Antes de comenzar repasemos los requisitos mínimos de Rocks 4.3:

- Para el *frontend*:
  - 20 GB de disco duro.
  - 1 GB de RAM para arquitectura x86_64.
  - 2 interfaces de red.
- Para los nodos:
  - 20 GB de disco duro.
  - 512 MB de RAM.
  - 1 interfaz de red.

Y solamente soporta arquitecturas x86, x86_64 y IA64.

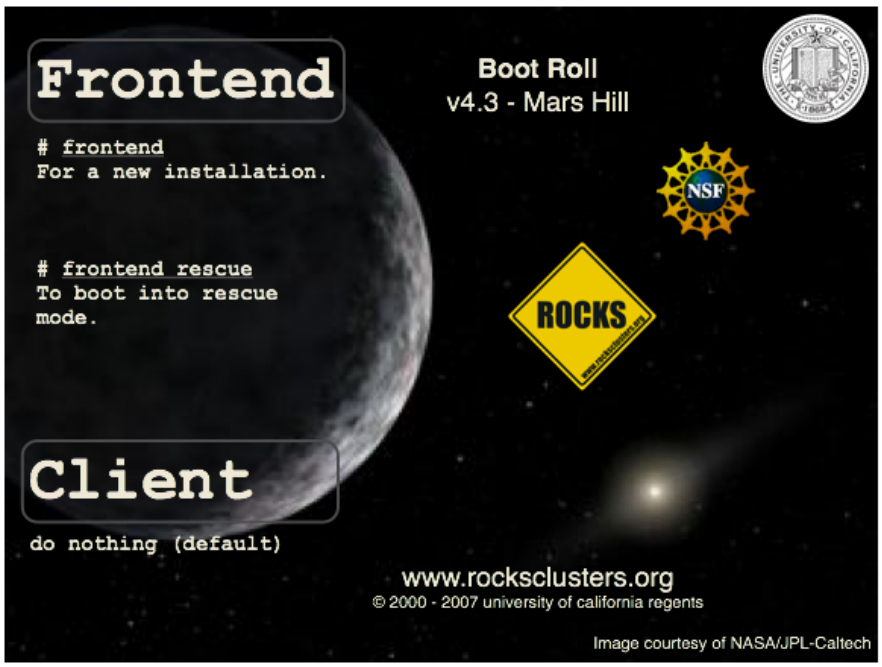



Figura B.1: Pantalla de inicio de instalación

## B.1. Instalación del nodo maestro

Se necesita el disco *kernel boot*, que podemos descargar de [7]. Una vez encendemos el nodo maestro, con el CD-ROM o DVD-ROM, muestra la figura B.1.

Para comenzar con la instalación debemos escribir:

```
frontend
```

Y la instalación del nodo maestro comenzará, simplemente deberemos seguir las instrucciones que encontremos para completar la instalación.

Vemos en la figura B.2, los botones de CD/DVD-based Roll y Download con los que podremos añadir las Rolls que consideremos necesario para instalar Rocks. Es en este punto, donde podremos insertar el CD/DVD de cualquier Roll que queramos instalar, encontramos las dos pantallas que podemos ver en las figuras B.3 y B.4.

Una vez añadido todo el software que necesite el cluster, tendremos una pantalla parecida a la que mostramos en la figura B.5.

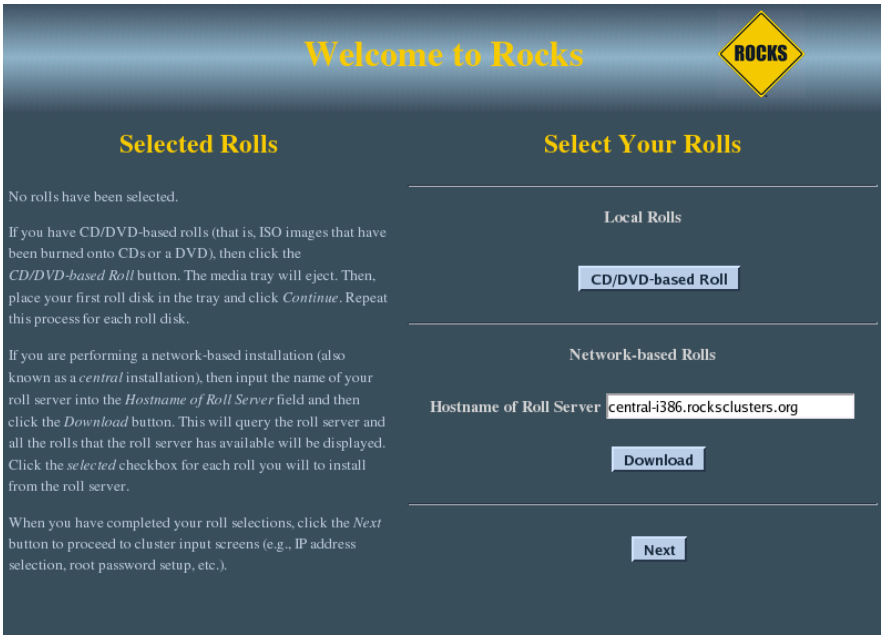



Figura B.2: Pantalla de selección de Rolls

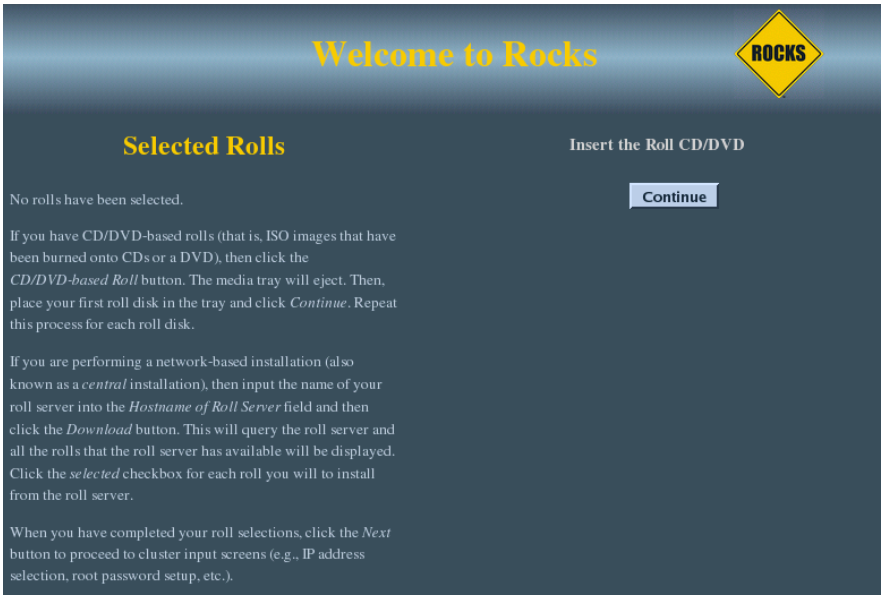



Figura B.3: Pantalla para insertar CD/DVD

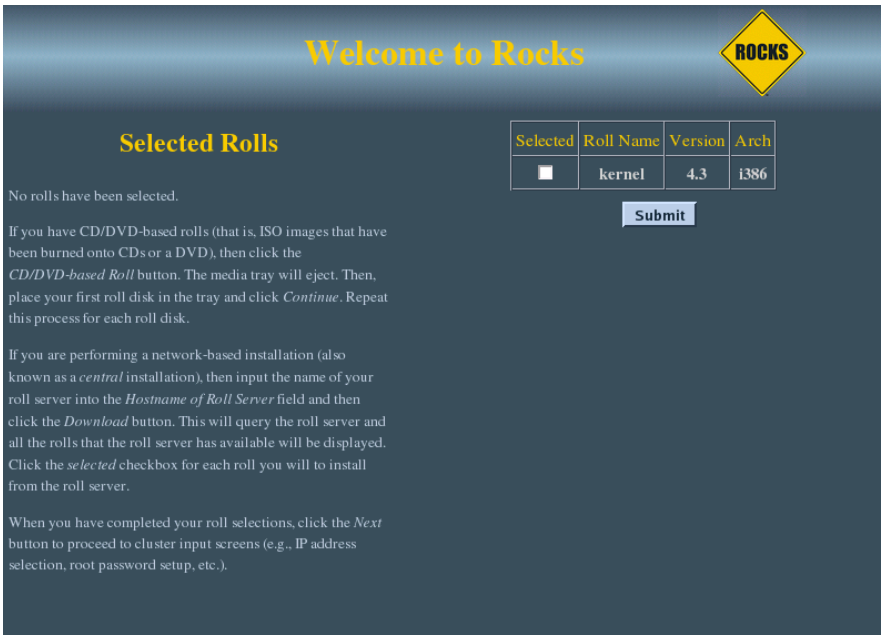



Figura B.4: Pantalla de selección de Roll

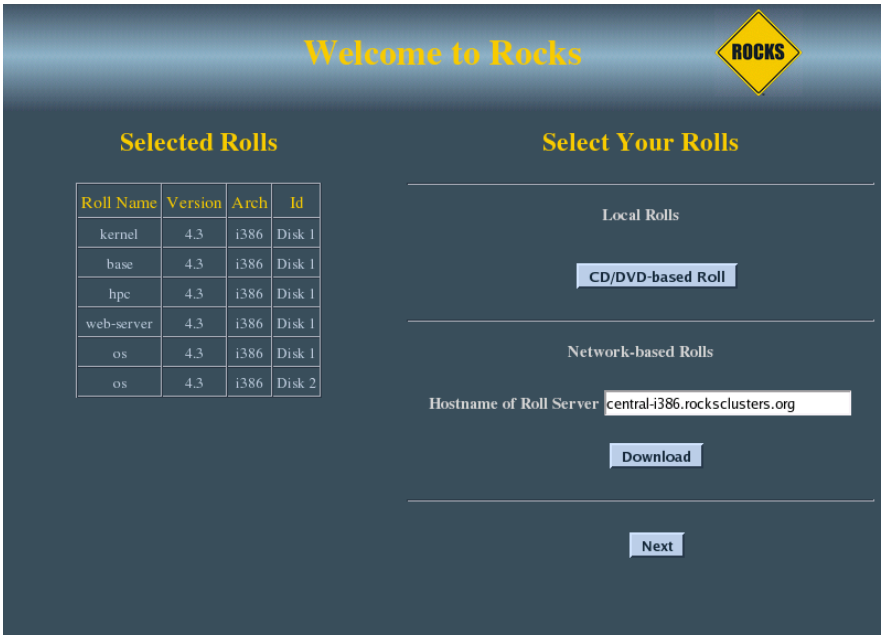



Figura B.5: Pantalla con todas las Rolls


Figura B.6: Pantalla de "Fully-Qualified Host Name"

En las siguientes pantallas nos pedirá información del cluster sólo *Fully-Qualified Host Name* (figura B.6) es obligatorio, el resto de campos (*Cluster Name*, *Certificate Organization*, *Certificate Locality*, *Certificate State*, *Certificate Country*, *Contact*, *URL* y *Latitude/Longitud*) son opcionales. No obstante recomendamos rellenar al menos el campo de *Contact* con el email del administrador.

Para las pantallas de configuración de red, deberemos tener en consideración la figura 2.5 del primer capítulo.

La pantalla B.7 configura el interfaz eth0, que es aquel que conecta los nodos esclavos, por defecto es 10.1.1.1 y la máscara de red 255.0.0.0.

Después vienen dos pantallas, figuras B.8 y B.9, para configurar el interfaz eth1, que es el interfaz que tiene salida a internet. Se necesitará la dirección IP de la máquina, el DNS, la máscara de red y el gateway.

A continuación introducimos la contraseña de root, y el servidor de tiempo. Llegando a la pantalla de particionamiento mostrada en la figura B.10.

Los tamaños por defecto son:

- / 8GB

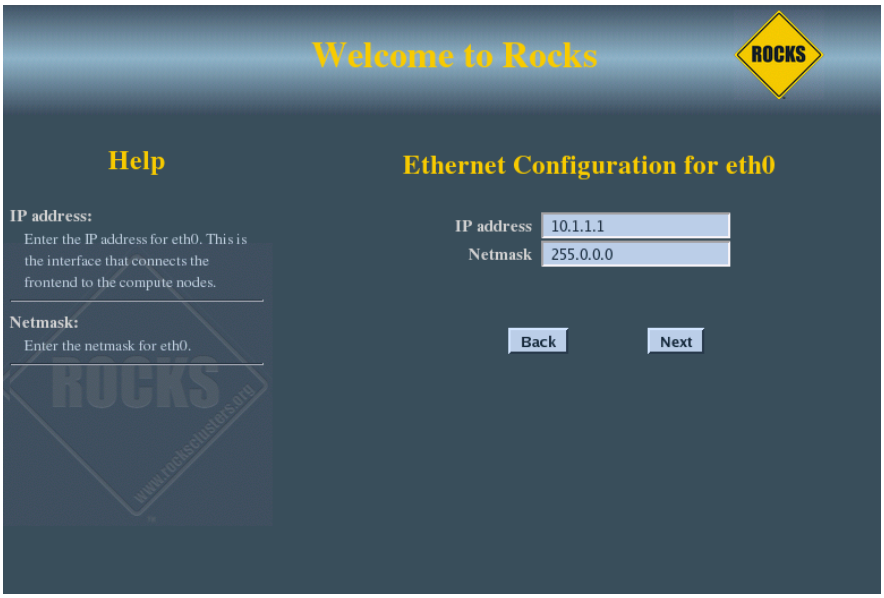



Figura B.7: Pantalla de Configuración de eth0

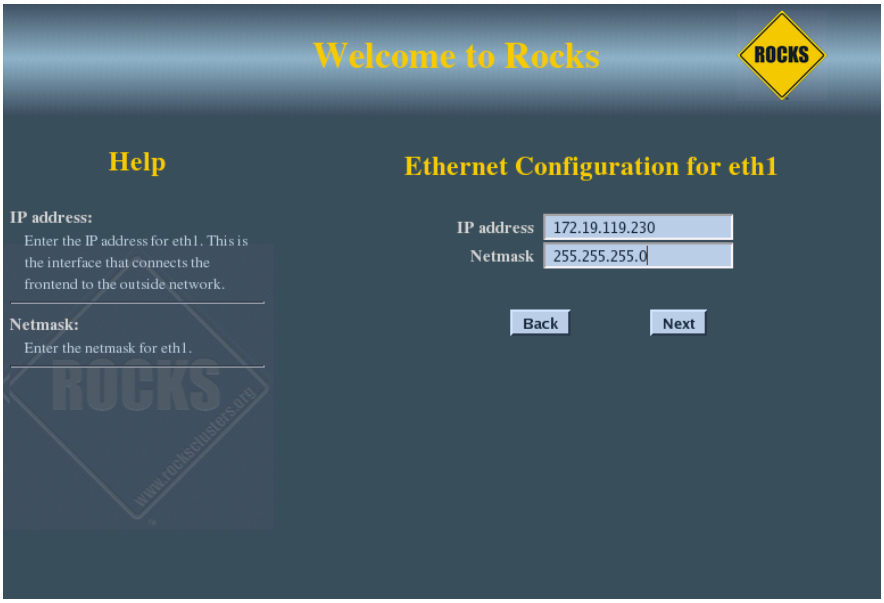



Figura B.8: Pantalla de Configuración de eth1

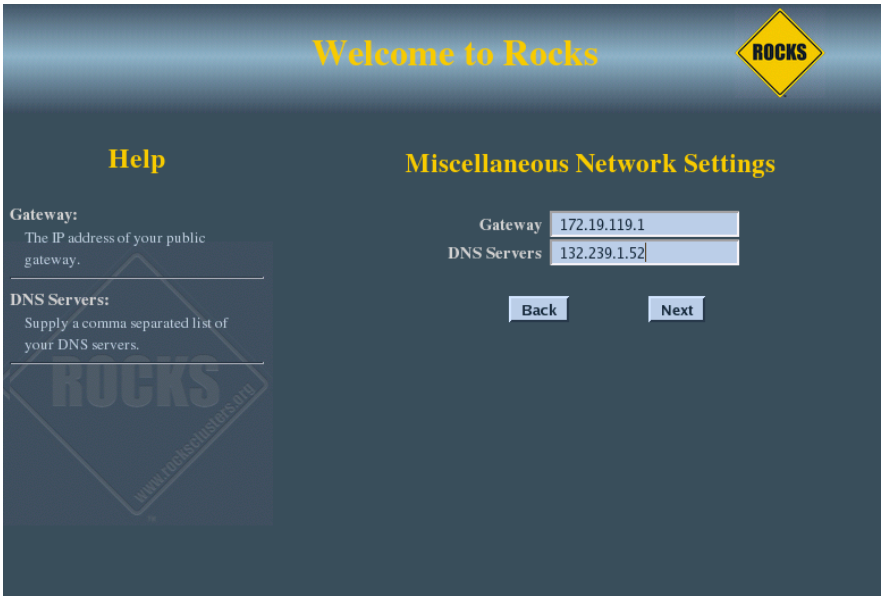



Figura B.9: Pantalla de Configuración de eth1

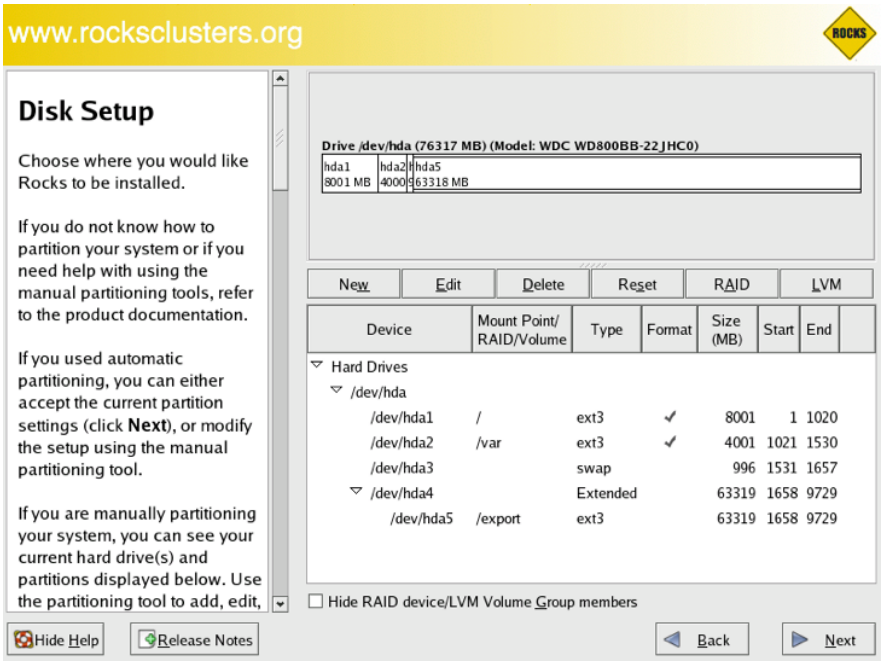



Figura B.10: Pantalla de particionamiento

- /var 4GB
- swap 1GB
- /export el resto. Es importante destacar que Rocks cuelga de aquí los directorios home y todos aquellos que vayan a ser compartidos por red.

### B.1.1. Directorios home

Rocks está orientado a SSI (*Single System Image*), y una de las formas de conseguir ésto es que al iniciar sesión monta automáticamente los directorios `home` tanto en el maestro, como en los esclavos. Esto es fundamental para ejecutar los programas en paralelo, sin necesidad de sincronizar los resultados, dando una única imagen. La contrapartida consiste tener algunas cosas en cuenta:

Los directorios del home residen en */state/partition1/* y son montados al iniciar sesión en */home*. Existe un softlink entre */state/partition1* (que es una partición del HD) y */export/*.

Si vamos al directorio */export/home* descubrimos un usuario *install*, viene a ser un "usuario de sistema". Este usuario se encarga de mantener nuestra distribución de Rocks, que esta formada por las Rolls, los rpm que tenemos instalados y la distribución inicial. No podremos logearnos como install, lógicamente, y sólo el administrador podrá modificar el contenido de este directorio.

Como consecuencia de la política de montaje de directorios, al crear una cuenta de usuario, éste debera hacer "login" en el maestro, para crear los diversos ficheros de usuario necesarios. El motivo es que la información de usuario reside en el maestro o *frontend*, y al iniciar sesión en un nodo se exporta.

Dado que es un tema que resulta confuso, mencionaremos los dos mecanismos de montaje que posee Rocks:

- Mount: Rocks monta los ficheros del usuario por NFS en el nodo donde se conecta. Es muy importante para que los cambios realizados estén sincronizados.
- Automount: Rocks usa autofs para que todo sea automático, de forma que al iniciar sesión lo monta y lo desmonta cuando el usuario deja el sistema.

Ahora sólo queda insertar los CDs/DVD según nos diga y se instalará.


Figura B.11: Instalación

## B.2. Instalación de los nodos esclavos

Para realizar la instalación de todos los esclavos, éstos han de estar conectados a la red de esclavos (no tienen acceso a internet y ven el interfaz eth0 del maestro). Es fundamental que el nodo tenga arranque por red como primera opción de la BIOS.

Una vez están conectados, se inicia sesión como root en el maestro y se teclea:

```
insert-ethers
```

Llegamos a una pantalla como la mostrada en la figura B.12.

Y una vez seleccionado el tipo de dispositivo que vamos a añadir a la red, Rocks se encarga de introducirlo en la base de datos MySQL e iniciar la instalación, ya sea de un nodo, de un NAS u otro dispositivo. Sólo destacar que necesita leer las peticiones y respuestas DNS que se generan al arrancar el equipo, deberemos tenerlo en consideración.

En las figuras B.13, B.14 y B.15, vemos cómo descubre el nodo. Automáticamente lo inserta en la base de datos y comienza la instalación.

Si la instalación falla, o se cuelga, deberemos reiniciar los siguientes servicios:

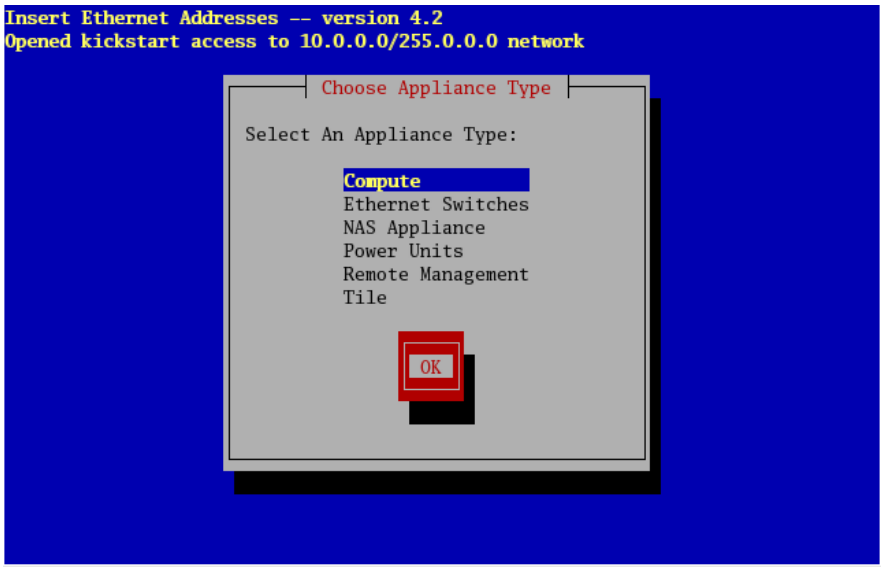



Figura B.12: Selección del tipo de nodos

```
# /etc/rc.d/init.d/httpd restart
# /etc/rc.d/init.d/mysqld restart
# /etc/rc.d/init.d/autofs restart
```

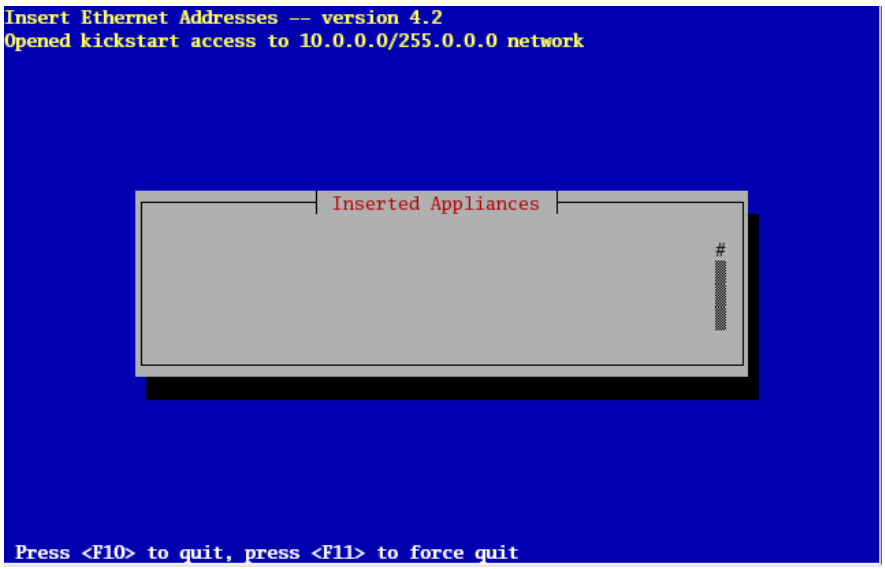



Figura B.13: Selección del tipo de nodos

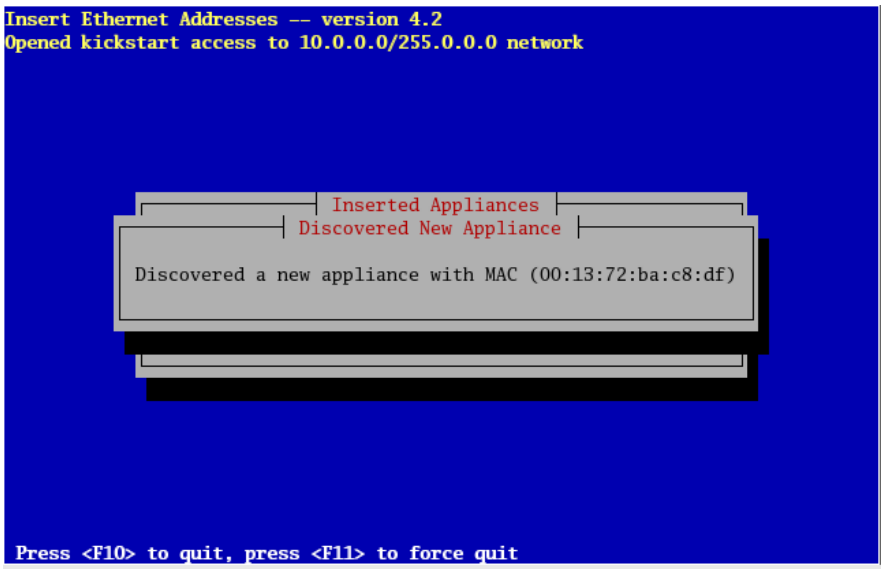



Figura B.14: Selección del tipo de nodos

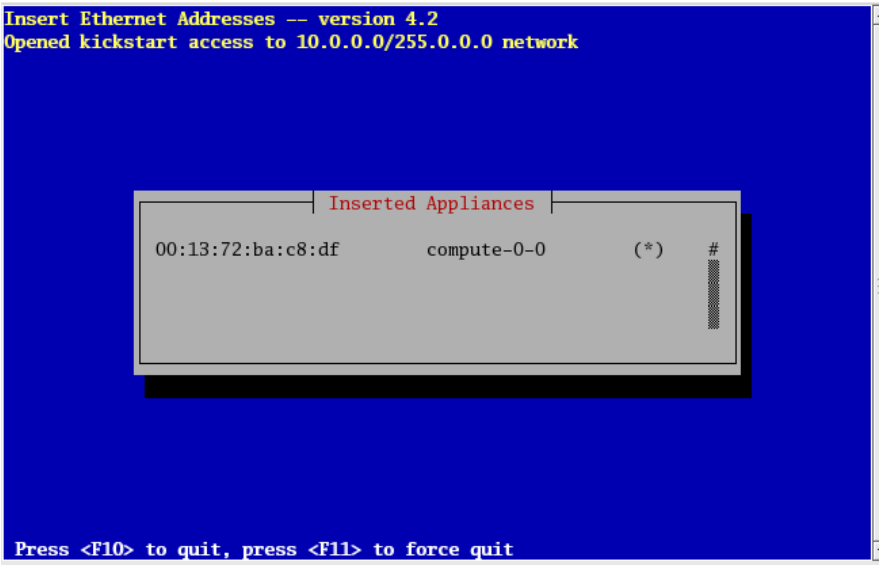



Figura B.15: Selección del tipo de nodos

# Apéndice C

# Sun Grid Engine

## C.1. Entorno

Podemos ver este punto con detalle en [24] y [2]; no obstante en el presente anexo describiremos brevemente el funcionamiento de SGE.

Se recomienda instalar esta Roll con el *frontend* desde cero, ya que, la integración en un frontend instalado resulta complicada e inestrable. Al realizar la instalación desde cero obtendremos automáticamente un entorno SGE estable. Para comprobarlo podemos ver la salida de los siguentes comandos:

```
[sysadm1@frontend-0 sysadm1]$ echo $SGE_ROOT
/opt/gridengine
[sysadm1@frontend-0 sysadm1]$ which qsub
/opt/gridengine/bin/glinux/qsub
```

## C.2. Mandar tareas

Recomendamos generar scripts para la ejecución de las tareas, ya que, si aún no estamos familiarizados con los comandos resulta cómodo y efectivo. La generación de scripts sigue una estructura de cabecera y comandos a ejecutar:

```
#!/bin/bash
##File cat sleep.sh
#$ -cwd
#$ -j y
#$ -S /bin/bash
#
date
sleep 10
date
```

Los parámetros del script anterior se traducen en:

- La línea *-cwd* sirve para indicar que la referencia al path se realiza en el directorio actual (SGE se encarga de gestionar los paths).
- La línea *-j y* sirve para que los errores vayan a la misma salida que el resultado de ejecución del programa (generalmente son diferentes). Esto nos puede resultar muy útil para la ejecución remota, debido a que SGE guarda esta salida en un fichero que podremos leer una vez terminada la ejecución de la tarea.
- La línea *-S /bin/bash* indica que el interprete de comandos es bash.

Para lanzar las tareas usamos el comando *qsub script.sh* con lo que añadimos una tarea a la cola de trabajo. Veámoslo a través de un ejemplo:

```
qsub script.sh
Your job 17 ("script.sh") has been submitted
```

SGE nos permite ver el estado de ejecución de la tarea con `qstat`, generalmente seguido de `-f` para ver todas las tareas de la cola o `-j jobnumber` para ver el estado detallado de una tarea. A continuación un ejemplo:

```
[nacho@ash25 ~]$ qstat -f
queuename      qtype used/tot. load_avg arch    states
---------------------------------------------------------------------------
all.q          BIP   0/2        0.00     lx26-amd64
[nacho@ash25 ~]$  qstat -j 19
==========================================================================
job_number:                 19
exec_file:                  job_scripts/19
submission_time:            Sun May 18 17:10:40 2008
owner:                      nacho
uid:                        500
group:                      nacho
gid:                        500
sge_o_home:                 /home/nacho
sge_o_log_name:             nacho
sge_o_path:                 /opt/gridengine/bin/\
                            lx26-amd64:/..
sge_o_shell:                /bin/bash
sge_o_workdir:              /home/nacho/sge
sge_o_host:                 ash25
account:                    sge
cwd:                        /home/nacho/sge
path_aliases:               /tmp_mnt/ * * /
merge:                      y
mail_list:                  nacho@ash25.tsc.uc3m.es
notify:                     FALSE
job_name:                   sleep.sh
jobshare:                   0
shell_list:                 /bin/bash
env_list:
script_file:                sleep.sh
scheduling info:            There are no messages
                            available
```

**Fichero C.1: Plantilla ejemplo para SGE**

```
#!/bin/bash
#
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -M email@domain
#$ -m e
#$ -pe mpi 4-8
#$ -o output/hello.out
#$ -e output/hello.err
#$ -V
#$ -q all.q
#$ -l h_rt=0:1:0
mpiexec -machinefile $TMPDIR/machines -np $NSLOTS ./program
```

**Fichero C.2: Funcionamiento de ejecución paramétrica.**

```
#!/bin/bash
#
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -t 1-3
programa -fin matrix_size$SGE_TASK_ID
```

Para parar el proceso usamos `qhold -j jobnumber`. Es decir paramos la ejecución y se queda en estado de espera. Si lo que queremos es eliminar el proceso de la cola, `qdel jobnumber` elimina de la cola la tarea jobnumber (número de la tarea)

```
qdel 18
nacho has registered the job 18 for deletion
```

Para mandar tareas que necesiten varios nodos, es decir aplicaciones paralelas, deberemos cambiar ligeramente la estructura de nuestra plantilla (fichero C.1).

En este caso es importante saber que las variables `TMPDIR` y `NSLOTS` son variables de SGE, y deberemos usarlas, pues ejecutamos en el entorno que SGE nos proporciona. Es decir, SGE nos proporciona unos slots asociados a diversas máquinas, y si cambiamos estos parametros es muy probable que no ejecutemos correctamente la aplicación.

Además de ejecutar en paralelo SGE cuenta con ejecución paramétrica en lo que se denomina `Job Arrays`, en el ejemplo que encontramos en el fichero C.2, podemos ver como utilizarlo, la variable `SGE_TASK_ID` toma valores siguiendo el parámetro `-t`, lanzando tres trabajos, con distintos parámetros: `matrix_size1`, `matrix_size2` y `matrix_size3`.

### C.2.1. Opciones generales

- `-cwd`: Lanza la tarea en el directorio donde es lanzada. Por defecto se sitúan en el `$HOME` de cada usuario.
- `-pe`: *pararel enviroment* indica que recursos necesita inicializar.
- `-S [shell]`: Campo obligatorio, indica el interprete de comandos.
- `-M mail@addr.ess`: Esta opción indica que se mande un correo a la dirección indicada al finalizar.
- `-N` <`req_name`> Para indicar el nombre de la petición por defecto es el nombre del script.
- `-o` <`filename`> Indica el fichero de salida que por defecto es el nombre del script y termina en .o[jobnumber].
- `-e` <`filename`> Indica el fichero de errores del programa, que por defecto es el del script terminado en .e[jobnumber].
- `-i [host:/directorio/]`<`filename` > Indica el fichero de entrada del programa que sustituye a la entrada estándar.
- `-j y` Hace que los errores tengan la misma salida que el programa (`-e` es ignorado).
- `-v var1[=val1][,var2[=val2],...]` Carga variables de entorno. Es recomendable cargarlas en el script y no por parametros.
- `-V` Carga todas las variables actuales.
- `-h` Encola un trabajo en estado de parada. Con `qaltar -hU` lo "lanzas".
- `-l h_rt=x:y:z` Limita el tiempo de ejecución a x horas, y minutos y z segundos
- `-l mf=`<`memory`> Especifica los requisitos de memoria sin los cuales no se ejecutara la tarea.
- `-q` <`queue_name`> Especifica la cola a usar para esa tarea.
- `-q` <`queue_name`>`@`<`nodename`> Especifica que la tarea será lanzada en <`nodename`> de la cola <`queue_name`>.

## C.3. Colas

Para manejar colas tenemos que saber que existen 5 tipos de usuarios:

- Managers: pueden crear colas y lanzar procesos. Poseen todos los permisos
- Executers: poseen todos los permisos salgo para los comandos `qmon` y `qconf` donde sólo podrán modificar todo aquello que no sea de sistema.
- Owner: tienen plenos permisos sobre las tareas y colas propias, pero sólo podrán ver y listar las ajenas. Podrán ver la configuración pero no modificarla.
- Users: tienen los mismos permisos que Owner, pero el comando `qacct` esta restringido a una tarea.

En nuestro caso tenemos tres colas, `all.q` con todos los nodos, single.q con todos los nodos cuyos procesadores sólo poseen un core, y `quad.q` con aquellos nodos que cuentan con cuatro cores por procesador.

Es necesario marcar en `all.q` que `quad.q` y `single.q` son colas subordinadas, la forma más simple de hacerlo es:

```
qconf -mq <cluster_queue>
```

Al introducirlo presenta un menú donde podremos realizar las modificaciones. Aunque también podemos hacerlo desde qmon, en la solapa de subordinates, podemos añadir a la lista las deseadas.

Conceptualmente una lista subordinada, deja de funcionar cuando se ocupan determinado número de slots, en nuestro caso hemos configurado en all.q que single.q deje de funcionar cuando se llenen 14 slots en all.q y quad.q deje de funcionar cuando all.q tenga 32 slots ocupados. Esto es una aproximación, pues podemos jugar con la distribución de las tareas en all.q y en quad.q (o single.q) y enviar más trabajos que los nodos pueden mánejar, no obstante de esta manera nos aseguramos que no tengan más de 2 trabajos/core, podríamos hacer cuentas de capacidad, pero la realidad es que si necesitamos ejecutar en all.q, será porque el trabajo en cuestión requiere el mayor número de procesadores posibles, así pues, queda justificado el número de slots elegido.

Para terminar debemos mencionar que existen otros comandos que pueden ser de utilidad como `qhost` que muestra los nodos activos donde se ejecutan las tareas, veamos un ejemplo:

```
[nacho@ash25 ~]$ qhost
HOSTNAME                ARCH         NCPU  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------
global                  -               -     -       -       -       -       -
compute-0-0             lx26-amd64      2  0.00    5.8G  110.3M  996.2M     0.0
compute-0-1             lx26-amd64      2  0.00    5.8G  146.1M  996.2M     0.0
compute-0-10            lx26-amd64      8  0.00   31.4G  136.7M  996.2M     0.0
compute-0-11            lx26-amd64      8  0.05   31.4G  135.3M  996.2M     0.0
compute-0-12            lx26-amd64      2  0.00    5.8G  117.2M  996.2M     0.0
compute-0-13            lx26-amd64      2  0.00    5.8G  114.7M  996.2M     0.0
compute-0-2             lx26-amd64      2  0.00    5.8G  144.1M  996.2M     0.0
compute-0-3             lx26-amd64      2  0.00    5.8G  110.8M  996.2M     0.0
compute-0-4             lx26-amd64      2     -    7.8G       -  996.2M       -
compute-0-5             lx26-amd64      2  0.00    5.8G  117.0M  996.2M     0.0
compute-0-6             lx26-amd64      2  0.00    5.8G  110.3M  996.2M     0.0
compute-0-8             lx26-amd64      8  0.00   31.4G  128.9M  996.2M     0.0
compute-0-9             lx26-amd64      8  0.00   31.4G  133.9M  996.2M     0.0
```

En este caso compute-0-4 se está apagado.

# Apéndice D

# Modules

En el presente anexo presentamos brevemente el gestor de entornos *modules*, que consideramos una herramienta fundamental para controlar las variables de entorno dentro del cluster.

El funcionamiento es sencillo, posee unos comandos básicos que modifican las variables de entorno, en base a unos scripts en TCL. Seguiremos ese orden, presentaremos los comandos que proporciona, comprobaremos la modificación de las variables de entorno y por último presentaremos un script de configuración.

```
Modules Release 3.2.6 (Copyright GNU GPL v2 1991):
 Usage: module [switches][subcommand][subcommand-args]

Switches:
 -H|--help              this usage info
 -V|--version           modules version & configuration
                        options
 -f|--force             force active dependency resolution
 -t|--terse             terse format avail and list format
 -l|--long              long  format avail and list format
 -h|--human             readable format avail and list
                        format
 -v|--verbose           enable  verbose messages
 -s|--silent            disable verbose messages
 -c|--create            create caches for avail and
                        apropos
 -i|--icase             case insensitive
 -u|--userlvl <lvl>     set user level to
                        (nov[ice],exp[ert],adv[anced])
 Available SubCommands and Args:
 + add|load             modulefile [modulefile ...]
 + rm|unload            modulefile [modulefile ...]
 + switch|swap          [modulefile1] modulefile2
 + display|show         modulefile [modulefile ...]
 + avail                [modulefile [modulefile ...]]
 + use [-a|--append] dir [dir ...]
 + unuse                dir [dir ...]
 + update
 + refresh
```

```
+ purge
+ list
+ clear
+ help              [modulefile [modulefile ...]]
+ whatis            [modulefile [modulefile ...]]
+ apropos|keyword   string
+ initadd           modulefile [modulefile ...]
+ initprepend       modulefile [modulefile ...]
+ initrm            modulefile [modulefile ...]
+ initswitch        modulefile1 modulefile2
+ initlist
+ initclear
```

Podemos ver que los comandos más usados son:

```
module avail
module list
module add modulename
module rm modulename
```

Es decir, son los comandos principales porque listan los modulos disponibles, los que hemos añadido, añaden modulos y los eliminan. Un ejemplo de uso:

```
nacho@Nacho:~$ ssh ash25.tsc.uc3m.es
nacho@ash25.tsc.uc3m.es's password:
Last login: Sun Jan 18 23:13:29 2009 from XXXXXXXX
Rocks 4.3 (Mars Hill)
Profile built 16:01 27-Feb-2008

Kickstarted 18:12 27-Feb-2008
Rocks Frontend Node - ash25 Cluster
[nacho@ash25 ~]$ echo $PATH
/opt/gridengine/bin/lx26-amd64:/usr/kerberos/bin:/opt/gridengine/bin/lx26-
    amd64:/usr/java/jdk1.5.0_10/bin:/opt/globus/bin:/opt/globus/sbin:/opt/
    intel/clck/1.1:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/opt/eclipse:/
    opt/ganglia/bin:/opt/ganglia/sbin:/opt/maven/bin:/opt/rocks/bin:/opt/
    rocks/sbin:/home/nacho/bin
[nacho@ash25 ~]$ module avail

---------------- /share/apps/modulitos ----------------
icc             idb             ifc             impi
intel10         mkl             mpich_gnu       mpich_intel
mumps           mumps_seq       openmpi         petsc
totalview

---------- /usr/share/Modules/modulefiles ----------
dot             module-cvs      module-info     modules         null
use.own
[nacho@ash25 ~]$ module add intel10
[nacho@ash25 ~]$ echo $PATH
/opt/intel/fce/10.1.011//bin:/opt/intel/idbe/10.1.011/bin/bin:/opt/intel/cce
    /10.1.011//bin:/opt/gridengine/bin/lx26-amd64:/usr/kerberos/bin:/opt/
    gridengine/bin/lx26-amd64:/usr/java/jdk1.5.0_10/bin:/opt/globus/bin:/opt/
    globus/sbin:/opt/intel/clck/1.1:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/
    bin:/opt/eclipse:/opt/ganglia/bin:/opt/ganglia/sbin:/opt/maven/bin:/opt/
    rocks/bin:/opt/rocks/sbin:/home/nacho/bin
[nacho@ash25 ~]$ module list
Currently Loaded Modulefiles:
  1) icc      2) idb      3) ifc      4) mkl      5) intel10
[nacho@ash25 ~]$ module rm intel10
[nacho@ash25 ~]$ module list
No Modulefiles Currently Loaded.
```

```
[nacho@ash25 ~]$ echo $PATH
/opt/gridengine/bin/lx26-amd64:/usr/kerberos/bin:/opt/gridengine/bin/lx26-
    amd64:/usr/java/jdk1.5.0_10/bin:/opt/globus/bin:/opt/globus/sbin:/opt/
    intel/clck/1.1:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/opt/eclipse:/
    opt/ganglia/bin:/opt/ganglia/sbin:/opt/maven/bin:/opt/rocks/bin:/opt/
    rocks/sbin:/home/nacho/bins/sbin:/home/nacho/bin
```

En este caso el modulo intel10 añade los modulos icc, idb, ifc y mkl a su vez, con el fin de tener la misma version de compiladores de intel, y no tener que incluir uno a uno los cuatro modulos.

## D.1. Cómo hacer un modulo

En lugar de explicar detenidamente las diferentes formas de hacer módulos para añadirlos, hemos pensado que lo mejor sería plasmar un ejemplo sencillo donde podemos ver cómo se hace un módulo y los elementos indispensables que tiene. En el fichero D.1, podemos ver como generamos la ayuda en la función `ModulesHelp`. Pero lo más importante es la parte inferior, con el comandos `prepend-path` (y `setenv`), pues son estos comandos los que modificarán (o añadirán) la variable de entorno que le sigue. Si visitamos [3], podremos encontrar guias más completa.

**Fichero D.1: módulo para PETSc.**

```
#%Module1.0#####################################################
##
## modules modulefile
##
## modulefiles/petsc.  Generated from modules.in by
## root.
##
set name  PETSc

proc ModulesHelp { } {
global name
append str1 "\t" $name " module"
append str2 "\n\tThis adds " $name " to the"
        puts stderr $str1
        puts stderr $str2
        puts stderr "\tenvironment variables."
}

append str3  "loads the " $name " environment"
module-whatis   $str3

# for Tcl script use only
set     version         2.3.2
set     dir             /opt/petsc/petsc-2.3.2-p10

setenv          PETSC_DIR       $dir
prepend-path    PATH            $dir
prepend-path    PATH            $dir/bin
prepend-path    PATH            $dir/bmake
prepend-path    PATH            $dir/include
```

# Apéndice E

# Roll Restore

Esta Roll, generada con ayuda de las herramientas de Rocks, posibilita recuperar un determinado estado del maestro y, por tanto, del cluster completo. Constituye un backup de estado del nodo maestro. Su utilidad principal es ante problemas de hardware que obliguen a una instalación desde cero del nodo maestro. También para la migración a una versión superior de Rocks.

## E.1. Procedimiento

Para generar el CD/DVD que contiene nuestro Roll Restore, nos dirigimos al siguiente directorio:

```
cd /export/site-roll/rocks/src/roll/restore
```

Y simplemente introducimos:

```
make roll
```

Automáticamente se generará una imagen de CD/DVD de nuestro Roll Restore. Generalmente al instalar está Roll, en esta versión de Rocks, deberemos introducir a mano las particiones del *frontend*, a continuación recordamos las particiones y tamaños por defecto:

- / 8GB
- /var 4GB

- swap 1GB
- /export el resto. Es importante destacar que Rocks cuelga de aquí los directorios home y todos aquellos que vayan a ser compartidos por red.

## E.2. Añadir archivos al Roll Restore

El mecanismo principal de la Roll Restore consiste en un conjunto de makefiles, que evaluan las expresiones de ciertos ficheros. Encontraremos la configuración de los archivos de usuario, que se añaden automáticamente a la Roll, en:

```
/export/site-roll/rocks/src/roll/restore/version.mk
```

Y la de archivos de sistema:

```
/export/site-roll/rocks/src/roll/restore/src/system-files/version.mk
```

Lo mostramos en el ejemplo, basado en un ejemplo encontrado de [7], fichero E.1. Incluye ficheros vistos en otros capítulos, como por ejemplo las líneas 33 y 34, que incluiyen los módulos de IMPItool en el kernel del sistema.

**Fichero E.1: Ejemplo de backup de archivos de sistema.**

```
NAME    = restore-system-files
RELEASE = 1
#
# these are the files that will be restored when this roll is supplied
# during installation
#
FILES   = /etc/passwd /etc/shadow /etc/gshadow /etc/group \
          /etc/exports /etc/auto.home /etc/motd
# Save yourself
FILES  += /export/site-roll/rocks/src/roll/restore/src/system-files/version.
    mk
# Save contributed RPMs. WARNING: if they include a kernel RPM it may be
# installed instead of the kernel RPM you expect.  This was true for a
# CentOSplus kernel I needed only for a NAS node.  So this will happen for a
# re-install, but might not happen for an upgrade (it should take the newest
# kernel RPM, right?).
FILES  += /home/install/contrib/4.3/x86_64/RPMS/*
FILES  += /home/install/contrib/4.3/x86_64/SRPMS/*
# X-windows settings
#FILES  += /etc/X11/xorg.conf /etc/skel/.Xdefaults
# Customizations for the shells (csh, tcsh, bash)
FILES  += /etc/skel/.*
# Profile files, Modules
FILES  += /etc/profile.d/modules.csh /etc/profile.d/modules.sh \
          /state/partition1/apps/modules/*
# System-wide mail file
FILES  += /etc/postfix/main.cf /etc/postfix/transport /etc/aliases
# Local mail and crontabs for all users
FILES  += /var/spool/cron/* /var/spool/mail/*
# Any customized autofs maps
FILES  += /etc/auto.auto
# Any other customization
#As we saw on chapter two:
FILES += /etc/rc.modules
FILES += /var/411/FILES.mk
#wake on lan and ipmitool, again as we saw on chapter two:
FILES += /etc/init.d/wol /etc/init.d/ipmioverlan \
         /etc/ipmi_lan.conf
```

# Bibliografía


[1] R. F. Harrington, *Field Computation by Moment Methods*, IEEE Press, 1993.

[2] Sun Grid Engine Web page, `http://gridengine.sunsource.net/`.

[3] Modules Website, `http://modules.sourceforge.net/`.

[4] B. Callaghan, B. Pawlowski y P. Staubach, NFS Version 3 Protocol Specification, RFC 1813 (Informational), 1995.

[5] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler y D. Noveck, Network File System (NFS) version 4 Protocol, RFC 3530 (Proposed Standard), 2003.

[6] MPI, Message Passing Interface, Standard, `http://www.mpi-forum.org/`.

[7] Rocks Web page, `http://www.rocksclusters.org/`.

[8] The Official Warewulf/Perceus Portal, `http://www.perceus.org/portal/`.

[9] SCYLD Software, `http://www.scyld.com/scyld_os.html`.

[10] Score Cluster System Software, `http://freshmeat.net/projects/score/`.

[11] San Diego Supercomputer Center, `http://www.sdsc.edu/`.

[12] National Partnership for Advanced Computational Infrastructure, `http://www.npaci.edu/`.

[13] The phpMyAdmin Home Page, `http://www.phpmyadmin.net/home_page/index.php`.

[14] OpenMPI Website, `http://www.open-mpi.org/`.

[15] MPICH Home Page, `http://www.mcs.anl.gov/research/projects/mpi/mpich1/`.

[16] M. Eisler, XDR: External Data Representation Standard, RFC 4506 (Standard), 2006.

[17] RDDTool Website, http://oss.oetiker.ch/rrdtool/.

[18] TORQUE Resource Manager, http://www.clusterresources.com/products/torque-resource-manager.php.

[19] Intel Compilers Website, http://software.intel.com/en-us/intel-compilers/.

[20] *Intel Math Kernel Library Reference Manual*, 2009.

[21] Intel MPI Website, http://software.intel.com/en-us/intel-mpi-library/.

[22] Homepage of the Linux NIS/NIS+ Projects, http://www.linux-nis.org/index.html.

[23] The Postfix Home Page, http://www.postfix.org/.

[24] *Begginer's Guide to Sun Grid Engine 6.2*, 2008.

[25] GNU Compilers Website, http://gcc.gnu.org/.

[26] METIS Web Page, http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.

[27] MUMPS Web Page, http://mumps.enseeiht.fr/.

[28] Linpack Website, http://www.netlib.org/linpack/.

[29] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith y H. Zhang, PETSc Users Manual, Technical Report ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2008.

[30] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith y H. Zhang, PETSc Web page, http://www.mcs.anl.gov/petsc.

[31] Y. Zhang, M. Tailor, T. K.Sarkar, H. Moon y M. Yuan, *IEEE Antennas and Propagation Magazine* (2008).

[32] Top 500 Website, http://www.top500.org.

[33] BLAS Website, http://www.netlib.org/blas/.

[34] Y. Saad, *Iterative Methods for Sparse Linear Systems, 2nd edition*, Society for Industrial and Applied Mathematics, 2003.

[35] Matrix Market Web page, `http://math.nist.gov/MatrixMarket/`.

[36] Y. Zang y T. K. Sarkar, *Parallel Solution of Integral Equation-based EM Problems in the Frequency Domain*, John Wiley and Sons, 2009.

[37] ScaLAPACK Website, `http://www.netlib.org/scalapack/`.

[38] PLAPACK Website, `http://www.cs.utexas.edu/~plapack/`.

[39] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, Society for Industrial and Applied Mathematics, 1995.

[40] L. E. García-Castillo, An Introduction to Computacional Electromagnetics.

[41] R. Mecklenburg, *Managing Projects with GNU Make*, O'Reilly, 2004.

[42] C. Newham y B. Rosenblatt, *Learning the bash Shell, Third Edition*, O'Reilly, 2005.

[43] T. Oetiker, H. Partl, I. Hyna y E. Schlegl, *The Not So Short Introduction to LaTeX 2ε*, Free Software Fundation, 2008.

[44] J. C. Adams, W. S. Brainerd, J. T. Martin, B. T. Martin y J. L. Wagener, *Fortran 90 Handbook*, McGraw-Hill, 1992.

[45] S. J. Chapman, *Fortran 95/2003 for Scientists & Engineers*, McGraw-Hill, 2007.